



AFRL-RI-RS-TR-2013-191

ADVANCED HOMOMORPHIC ENCRYPTION ITS APPLICATIONS AND DERIVATIVES (AHEAD)

IBM WATSON RESEARCH CENTER

SEPTEMBER 2013

FINAL TECHNICAL REPORT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

STINFO COPY

**AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09. This report is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (<http://www.dtic.mil>).

AFRL-RI-RS-TR-2013-191 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE DIRECTOR:

/ S /

CARL THOMAS
Work Unit Manager

/ S /

MARK LINDERMAN
Technical Advisor, Computing
& Communications Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE				Form Approved OMB No. 0704-0188	
<p>The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.</p> <p>PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.</p>					
1. REPORT DATE (DD-MM-YYYY) SEPTEMBER 2013		2. REPORT TYPE FINAL TECHNICAL REPORT		3. DATES COVERED (From - To) FEB 2011 – MAR 2013	
4. TITLE AND SUBTITLE ADVANCED HOMOMORPHIC ENCRYPTION ITS APPLICATIONS AND DERIVATIVES (AHEAD)				5a. CONTRACT NUMBER FA8750-11-C-0096	
				5b. GRANT NUMBER N/A	
				5c. PROGRAM ELEMENT NUMBER 62303E	
6. AUTHOR(S) Tal Rabin, Nigel Smart, Daniel Wichs, Craig Gentry, Zvika Brakerski, Stefano Tessaro, Shai Halevi, Dan Boneh, Victor Shoup, Daniele Micciancio, Mark Zhandry. Chris Peikert David Cash, Michelle Downes, Alptekin Kupcu				5d. PROJECT NUMBER AHEA	
				5e. TASK NUMBER DI	
				5f. WORK UNIT NUMBER BM	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) <div style="display: flex; justify-content: space-between;"> <div style="width: 45%;"> PRIME IBM Watson Research Center 1101 Kitchewan Road Yorktown Heights, NY 10598 </div> <div style="width: 45%;"> SUB Stanford University 340 Panama St Stanford, CA 64305 </div> </div>				8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) <div style="display: flex; justify-content: space-between;"> <div style="width: 45%;"> Defense Advanced Research Projects Agency 3701 North Fairfax Dr Arlington, VA 2203 </div> <div style="width: 45%;"> Air Force Research Laboratory/RITA 525 Brooks Road Rome NY 13441-4505 </div> </div>				10. SPONSOR/MONITOR'S ACRONYM(S) AFRL/RI	
				11. SPONSOR/MONITOR'S REPORT NUMBER AFRL-RI-RS-TR-2013-191	
12. DISTRIBUTION AVAILABILITY STATEMENT Approved for Public Release; Distribution Unlimited. This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09. See next page for more details.					
SUPPLEMENTARY NOTES					
14. ABSTRACT The ultimate goal of the Programming on Encrypted Data (PROCEED) program is to advance the study and design of flexible and efficient techniques for processing encrypted data, outsourcing computation and adding robustness to cryptographic computations. The IBM portion of PROCEED, Advanced Homomorphic Encryption its Applications and Derivatives (AHEAD) is documented in this report. The AHEAD team has developed and implemented new and improved protocols for computing on encrypted data, and a deepened understanding of the foundations of secure computation. This report describes the work performed by IBM Watson Research Center and the subcontractors on this effort, Stanford University and the University of California, San Diego from February 2011 through March 2013.					
15. SUBJECT TERMS Fully Homomorphic Encryption (FHE), Secure Multi-Party Computation (SMC), Cryptographic Computation					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT UU	18. NUMBER OF PAGES 452	19a. NAME OF RESPONSIBLE PERSON CARL THOMAS
a. REPORT U	b. ABSTRACT U	c. THIS PAGE U			19b. TELEPHONE NUMBER (Include area code) N/A

The program was cancelled due to budget constraints.

All papers cited are approved for public release.

Papers authored by IBM personnel or in collaboration with IBM personnel have previously been reviewed by DARPA Public Release Center.

Papers authored by University subcontractors with work conducted on campus at a University are Contracted Fundamental Research (CFR) and do not require public release reviews.

A list of each paper with the appropriate DISTAR Public Affairs Review number and date, or indication that it meets the criteria for contracted Fundamental Research exclusion from public affairs review is below:

Public Affairs Approval of papers in Appendix

1. Fully Homomorphic Encryption without Squashing Using Depth-3 Arithmetic Circuits.
Approved for Public Release, DISTAR Case # 18107, October 25, 2011

2. Fully Homomorphic Encryption without Bootstrapping.
Approved for Public Release, DISTAR Case #17837, August 8, 2011

3. Targeted Malleability: Homomorphic Encryption for Restricted Computations.
Approved for Public Release, Contracted Fundamental Research

4. Trapdoors for Lattices: Simpler, Tighter, Faster, Smaller.
Approved for Public Release, Contracted Fundamental Research

5. Homomorphic Evaluation of the AES Circuit.
Approved for Public Release, DISTAR Case # 19368, June 11, 2012

6. Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP.
Approved for Public Release, Contracted Fundamental Research

7. When Homomorphism Becomes a Liability.
Approved for Public Release, Contracted Fundamental Research

8. Quantum-Secure Message Authentication Codes.
Approved for Public Release, Contracted Fundamental Research

9. Dynamic Proofs of Retreivability via Oblivious RAM.
Approved for Public Release, DISTAR Case # 19972, October 1, 2012

10. Hardness of SIS and LWE with Small Parameters.
Approved for Public Release, Contracted Fundamental Research

11. How to Delegate Secure Multiparty Computation to the Cloud.
Approved for Public Release, Contracted Fundamental Research

12. An Equational Approach to Secure Multi-Party Computation.
Approved for Public Release, Contracted Fundamental Research
13. Semantic Security for the Wiretap Channel.
Approved for Public Release, Contracted Fundamental Research
14. Multi-Instance Security and its Application to Password-Based Cryptography.
Approved for Public Release, Contracted Fundamental Research
15. To Hash or Not to Hash Again? (In)differentiability Results for H^2 and HMAC.
Approved for Public Release, Contracted Fundamental Research
16. FHE Library - Design and Implementation of a Homomorphic-Encryption Library
Approved for Public Release, DISTAR Case # 20493, January 18, 2013
17. Using Homomorphic Encryption for Large Scale Statistical Analysis
Approved for Public Release, Contracted Fundamental Research

Table of Contents

1.0	Executive Summary	1
2.0	Introduction and Program Overview	2
2.1	Background	2
2.1.1	The PROCEED PROGRAM.....	2
2.1.2	The AHEAD PROJECT	3
3.0	Program Work: Methods and Assumptions	4
3.1	Work Plans and Methods	4
3.1.1	TA 2: Foundations of secure computations.....	4
3.1.2	TA 3: Foundations of supporting security technologies	4
3.1.3	TA 4: Implementing fully homomorphic encryption	4
3.2	Deliverables and Assumptions	4
4.0	Accomplishments and Results.....	6
4.1	Summary of Major Contributions	6
4.2	Technology Transitions and Deliverables	7
4.3	Publications	9
5.0	Conclusions and Recommendations	19
6.0	References	1:
7.0	Appendix	42
8.0	List of Symbols, Abbreviations, and Acronyms	43
9.0	Public Release Approvals for Papers in Appendix	24

1.0 Executive Summary

The ultimate goal of the PROCEED AHEAD project was to advance the study and design of flexible and efficient techniques for processing encrypted data, outsourcing computation and adding robustness to cryptographic computations. Examples of problems that would benefit from such techniques are the verification of policy compliance on encrypted data, spam detection on encrypted data, efficient delegation of computation and more. The contributions of this program are already and will continue to improve the ability to process and manage data, in particular encrypted data, in a robust and fully functional way without sacrificing the confidentiality of information and the privacy of users.

We have provided an implementation of Fully Homomorphic Encryption (FHE). The main goal of the implementation was to help us evaluate the different algorithmic approaches, and to innovate on implementation techniques to bypass bottlenecks in the proposed schemes. We stress that building a usable implementation is not just about programming tricks, but rather mathematical innovations that improve the running times in practice. These improvements resulted in large constant factor speed-ups that are ignored in the asymptotic calculations of theoretical research papers, but are essential for a practical implementation. Our approach to making fully homomorphic encryption more efficient using speed-ups improved performance by several orders of magnitude. These improvements come from modifications to the basic schemes of Gentry and others that have little asymptotic impact, but have a large impact in practice.

An attractive approach to address issues of privacy is to resort to the area of secure multiparty computations (SMC) as extensively studied in cryptography. However previous solutions in this area are mostly theoretical, and are hardly used in practice due to their complexity. To address this we have proposed a novel framework for the design and analysis of secure computation protocols which allows for much simpler modeling and analysis of cryptographic protocols. Outsourcing computations has also previously suffered from the lack of efficient algorithms. In addition, we created algorithms for delegating computation and spam filters.

During just the first two years of the project the PROCEED AHEAD team has developed and implemented new and improved protocols for computing on encrypted data, and a deepened understanding of the foundations of secure computation. The report that follows describes the work of IBM, Stanford University, and University of California, San Diego on the PROCEED AHEAD project from February 2011 through March 2013.

2.0 Introduction and Program Overview

2.1 Background

Homomorphic Encryption is a form of encryption where a specified algebraic operation is performed on the plaintext and another (possibly different) algebraic operation is performed on the ciphertext. There are several forms of homomorphic encryption that allow an addition or multiplication operation on the plaintext, but to preserve the ring structure of plaintext both addition and multiplication operations must be supported. Using these methods, any circuit could be homomorphically evaluated, effectively allowing the construction of programs which may be run on encryptions of their inputs to produce an encryption of their output. Since the program would never decrypt its input, it could be run by an untrusted party, or transmitted over an untrusted media, without revealing its inputs or internal state.

The utility of this scheme is well known, but the algorithms and computation complexity of current implementations are burdensome. To be useful an efficient scheme must be developed and integrated into modern computing.

2.1.1 The PROCEED PROGRAM

PROgramming Computation on EncryptEd Data (PROCEED) is a program focused on creating practical methods for computing on encrypted data and is made up of the six Technical Areas (TA) listed below. The PROCEED AHEAD program which we report on here covers Technical Areas 2, 3, and 4 only.

- TA1. Mathematical Foundations of Fully Homomorphic Encryption
- TA2. Mathematical Foundations of Computation on Encrypted Data via Secure Multiparty Computation
- TA3. Mathematical Foundations of Supporting Security Technologies
- TA4. Implementation/Measurement/Optimization of Homomorphic Cryptography and Secure Multiparty Protocols
- TA5. Algorithms for Computation on Encrypted Data
- TA6. Programming Languages

The scope of the PROCEED effort is to design, develop, evaluate, integrate, demonstrate and deliver: new mathematical foundations for efficient secure multiparty computation; new mathematical foundations for efficient computation on encrypted data and supporting technologies/techniques; implementations of known and new schemes/protocols, measure and optimize these implementations; develop libraries of efficient algorithms and data structures; develop new programming languages and accompanying compilers and; provide input to the Integration Contractor for the development of a common Application Programmers Interface (API) and integration and evaluation of the research areas.

The PROCEED program efforts are broken up into four program phases as described below.

- ***Program Phase I Initial Capabilities.*** Phase I will focus on developing the initial capabilities for the mathematical foundations, measurement, algorithms for computation, and programming languages. An API will be developed and coordinated across all performers at the first Principal Investigator (PI) meeting.
- ***Program Phase II Alpha.*** Phase II will focus on development of alpha quality implementations of algorithms, optimized implementations, programming languages, an initial demonstration of remote regular expression matching and spam filter, and a refined definition of program metrics.
- ***Program Phase III Beta.*** Phase III will focus on the beta development of core algorithms, interoperability integration, and development of optimized implementations tested against defined program metrics.
- ***Program Phase IV Research Prototype.*** Phase IV will focus on the development of the research prototype and embedded application prototypes. Final demonstrations will include a functional spam filter prototype.

2.1.2 The AHEAD PROJECT

Advancing Homomorphic Encryption its Applications and Derivatives (AHEAD) is a sub-project within the broader PROCEED program focused on conducting research in Technical Areas 2, 3, and 4. The AHEAD research team has an extensive background in secure multiparty computation and homomorphic Encryption made up of IBM Research's Cryptography Group (Prime), and the Departments of Computer Science at Stanford University and the University of California, San Diego (UCSD). The AHEAD research team has worked to advance the study and design of flexible and efficient techniques for processing encrypted data, outsourcing computation and to add robustness to cryptographic computations including protection against accidental or malicious leakage of secret information. Examples of problems that will benefit from our work are the verification of policy compliance on encrypted data, spam detection on encrypted data, efficient delegation of computation, leakage resilient computation and more.

3.0 Program Work: Methods and Assumptions

3.1 Work Plans and Methods

We outline below the planned work to be accomplished within the three PROCEED program Technical Areas covered by AHEAD as defined at the onset of the program. Program work has been carried out jointly between IBM Research, Stanford University, and UCSD. In addition to the technical work, the AHEAD research team has participated in DARPA PI meetings and contributed to related DARPA events. All program results have been made available on the project website hosted by the PROCEED program integrator, Galois.

3.1.1 TA 2: Foundations of secure computations

- Programming models for secure computation. Extend Yao's garbled circuits to handle arithmetic functions efficiently. Design protocols for repeated executions and for programs with loops.
- Develop relations among different execution models and construct general transformations for transferring desirable protocol properties from one model to another.
- Design dedicated solutions for problems, e.g., pattern matching.

3.1.2 TA 3: Foundations of supporting security technologies

- Design homomorphic encryption for certain function families to enable restricted computation delegation.
- Verifying computation. Remove the need for FHE to delegate computation. Introduce proxy re-signatures in order to control malicious servers.
- Prevent side channel attacks using leakage resilience. Attempt to remove the reliance on leakage resilient hardware.

3.1.3 TA 4: Implementing fully homomorphic encryption

- Optimize Fully Homomorphic Encryption by speeding up key generation and encryption. Shrink the public key and ciphertext size.
- Explore fast two-party computation via fast Yao Circuits

3.2 Deliverables and Assumptions

Through the course of the planned four year PROCEED AHEAD program IBM, Stanford University and UCSD planned to deliver the following:

- An optimized implementation of fully homomorphic encryption.
- For all three tasks we delivered white papers and technical papers describing the results of the work.
- As we made progress on the theoretical underpinnings of these tasks we experimented with prototype implementations when appropriate.

Towards the end of the project we initiated technology transfer discussions with product groups within IBM and devoted resources to supporting a successful transition to real-world products. Completion criteria for all three tasks was planned to be the development of the required cryptographic systems along with theoretical optimizations and prototype software if appropriate.

The following is an outline of the planned schedule for the PROCEED AHEAD program. When the schedule for program work and deliverables was defined, it was assumed that IBM, Stanford, and USCD would all jointly contribute to program efforts for the duration of the planned four year program. This final report covers year one and year two contributions by the AHEAD team. At the close of year two IBM took leave of the program. Stanford and UCSD have continued their work under a new contract put in place with the Office of Naval Research and DARPA.

Year 1: Develop initial protocols for pattern matching on encrypted data. Begin investigating improved 2-party computation techniques of the tasks outlines in the proposal. Experiment with optimizations to our fully homomorphic encryption implementation.

Year 2: Continue developing our techniques for 2-party computation. Examine multiple executions of the same protocol in a multiparty setting. Tune protocols for pattern matching to the specific tasks of network guards and mail filtering on encrypted data. Continue experimenting with optimizations and modifications to the fully homomorphic system based on the results for year 1. Begin investigating the problem of delegating computation. Write and publish technical papers on intermediate results for all three tasks.

After Year 2 the program was cancelled. The Year 3 and 4 planned work is outlined below but was not completed.

Year 3: Build on our work from year 2 to and begin prototyping an application for our 2-party protocols, extend the investigation to include loops. Work on additional aspects of computation delegation. Use our optimized fully homomorphic encryption scheme for real world tasks such as network guards and other computations on encrypted data such as curve fitting (e.g., least-squares fit) on encrypted data. Initiate discussions with products within IBM to promote technology transfer. Submit additional papers for publication in leading conferences.

Year 4: Tune the research results to address the needs of product groups within IBM and to support technology transfer. As appropriate, release open source tools to enable other researchers to build on our work. Continue publishing technical papers on results of our research.

4.0 Accomplishments and Results

4.1 Summary of Major Contributions

IBM's work in PROCEED (in cooperation with others) significantly advanced the state-of-the-art in homomorphic encryption beyond the original blueprint of Gentry[1], taking it a big step toward practicality. First, the works of Gentry-Halevi[2], and Brakerski-Gentry-Vaikuntanathan[3], allow us to perform homomorphic computation without the need for squashing or even bootstrapping. Specifically the latter work provides much better handle on the growth of the noise during homomorphic computation, resulting in a significant speedup. Then the work of Gentry-Halevi-Smart[4] provides effective tools for working on many plaintext values at once, again resulting in significant speedups. Finally, the works of Gentry-Halevi-Smart[5], Brakerski[6], Gentry-Halevi-Smart[7], and Brakerski-Gentry-Halevi[8] provide different variants and optimizations that are likely to have additional practical advantages. The IBM FARTHER program administered through the Navy under PROCEED also contributed to these results.

Taken together, these works already provide roughly three orders of magnitude speedup over the Gentry-Halevi[9], results, allowing us to evaluate homomorphically circuits that were out of reach using only the Gentry '09 blueprint. For example, Gentry-Halevi-Smart[7], demonstrated that the AES-128 circuit can be evaluated homomorphically in under 36 hours. Since then we further optimized our code, and our current estimate is that we can do the same in just 3-4 hours.

UCSD has investigated the foundation and basic building blocks of lattice cryptography, used in the construction of some fully homomorphic encryption schemes and has published their results in two papers: "Trapdoor for Lattices: Simpler, Tighter, Faster, Smaller"[10] and "Hardness of SIS and LWE with Small Parameters"[11].

The first paper describes a new method for generating computationally hard lattices together with a trapdoor basis. The method is both much simpler and efficient, and produces better quality trapdoors than previous methods. The second paper studies the parameters for which the short integer solution (SIS) and learning with errors (LWE) problems are provably as hard as worst-case lattice problems. These are the two most fundamental problems used in all lattice cryptographic constructions, and using small parameters has clear efficiency benefits. The SIS problem is used in the construction of certain homomorphic hash functions, and it is shown that the modulus q used in SIS can be set almost as low as \sqrt{n} , (Previous work required $q > n$). The LWE problem is at the basis of the most recent fully homomorphic encryption schemes, and the parameter under investigation is the noise distribution. All previous work requires the noise to be at least \sqrt{n} , and to follow a Gaussian distribution. This can be undesirable, especially in the context of fully homomorphic encryption, because Gaussian distributions are harder to sample (than say, uniformly random strings) and because errors accumulate during the execution of homomorphic operations. So, larger noise rates results in reduced homomorphic capabilities. Our work shows that at least in some settings (when the number of LWE samples is sufficiently small) LWE is still hard when the noise is chosen uniformly at random, and with much smaller magnitude.

UCSD has also proposed a novel framework for the design and analysis of secure computation protocols in the work "An equational approach to secure multi-party computation"[12]. The main

feature of the framework is that it is fully asynchronous: local computations are independent of the relative ordering of messages coming from different communication channels. This allows for much simpler modeling and analysis of cryptographic protocols, which does not need a sequential ordering of all events. Besides making the formal proof of secure computation protocols more manageable, the framework has also potential efficiency benefits: as messages can be transmitted as soon as they can be computed (without compromising the security of the protocol), this may result in distributed protocols with lower latency. As a proof of concept, the paper analyzes two simple protocols, one for secure broadcast, and one for verifiable secret sharing, which demonstrate how the framework is capable to deal with probabilistic protocols, still in a simple and equational way.

Stanford's work has focused on different forms of homomorphic encryption and began with introducing a concept called "targeted malleability" which is designed to limit the homomorphic operations that can be done on encrypted data[13]. The primary motivation for this is to limit what can be done on ciphertexts. For example, a spam filter operating on encrypted data should only be allowed to run the spam predicate and nothing else. Several constructions for this concept have been provided.

Next, Stanford turned to optimizing fully homomorphic encryption. They first developed a variant of the BGV system that eliminates the need for the expensive modulus switching step[6]. This variant also enables us to use any modulus, including a power of 2, which can result in more efficient arithmetic. The resulting system has become known as Brakerski's FHE, named after the post-doc who developed it as part of the PROCEED program. Along the same lines Stanford also looked at a recent proposal for FHE due to Bogdanov and Lee which constructs an efficient FHE from coding theoretic assumptions[14]. They showed that the Bogdanov-Lee proposal is insecure, and in fact, any construction using their approach will be insecure[15].

Using these new FHE systems the team at Stanford built a prototype system that computes statistics on encrypted data, such as mean, standard deviation, and linear regression. The implementation is based on an optimized version of Brakerski's FHE that takes advantage of its arithmetic properties. Stanford also used large-scale batching to speed-up much of the computation. The resulting system can perform linear regression on moderate size encrypted data sets within a few hours on a single laptop. Parallelism can bring this down to a few minutes.

Finally, since the underlying mechanism behind FHE is based on hard problems on lattices, which are assumed to remain secure in the presence of quantum computers, Stanford looked at secure cryptographic primitives in the age of quantum computation. In particular, they built Message Authentication Codes that remains secure even when the devices using them are quantum[16]. One of the team's instantiations is a lattice-based MAC the presumably remains secure in a post-quantum settings.

4.2 Technology Transitions and Deliverables

The Stanford team developed a prototype system for performing statistical analysis on encrypted data. Their work focused on two tasks: computing the mean and variance of univariate and multivariate data as well as performing linear regression on a multidimensional, encrypted corpus. Due to the high overhead of homomorphic computation, previous implementations of

similar methods have been restricted to small datasets (on the order of a few hundred to a thousand elements) or data with low dimension (generally 1-4).

In this work[17], the Stanford team first constructed a working implementation of the scale-invariant leveled homomorphic encryption system of Brakerski. Then, by taking advantage of batched computation as well as a message encoding technique based on the Chinese Remainder Theorem, they showed that it becomes not only possible, but computationally feasible, to perform statistical analysis on encrypted datasets with over four million elements and dimension as high as 24. By using these methods along with some additional optimizations, the team was able to demonstrate the viability of using leveled homomorphic encryption for large scale statistical analysis.

The IBM team designed, implemented and delivered a Homomorphic Encryption (HE) software library[18] that implements the Brakerski-Gentry-Vaikuntanathan (BGV) homomorphic encryption scheme, along with many optimizations to make homomorphic evaluation runs faster, focusing mostly on effective use of the Smart-Vercauteren ciphertext packing techniques. Our library is written in C++ and uses the Number Theory Library (NTL) mathematical library. The NTL is a high-performance, portable C++ library providing data structures and algorithms for manipulating signed, arbitrary length integers, and for vectors, matrices, and polynomials over the integers and over finite fields (and can be found at <http://www.shoup.net/ntl>).

Very roughly, our HE library consists of four layers: in the bottom layer we have modules for implementing mathematical structures and various other utilities, the second layer implements our Double-CRT representation of polynomials, the third layer implements the cryptosystem itself (with the "native" plaintext space of binary polynomials), and the top layer provides interfaces for using the cryptosystem to operate on arrays of plaintext values. We think of the bottom two layers as the "math layers", and the top two layers as the "crypto layers", and describe them in detail in our work[18]. A block-diagram description of the library is given in Figure 1.

At the top level of the library we provide some interfaces that allow the application to manipulate arrays of plaintext values homomorphically. The arrays are translated to plaintext polynomials using the encoding/decoding routines and then encrypted and manipulated homomorphically using the lower-level interfaces from the crypto layer.

The basic operations that we have in the HE library scheme are the usual key-generation, encryption, and decryption, the homomorphic evaluation routines for addition, multiplication and automorphism (and also addition-of-constant and multiplication-by-constant), and the ciphertext maintenance operations of key-switching and modulus-switching.

In addition to the software described above the PROCEED AHEAD team has delivered many significant publications which are summarized in the Publications section of this report.

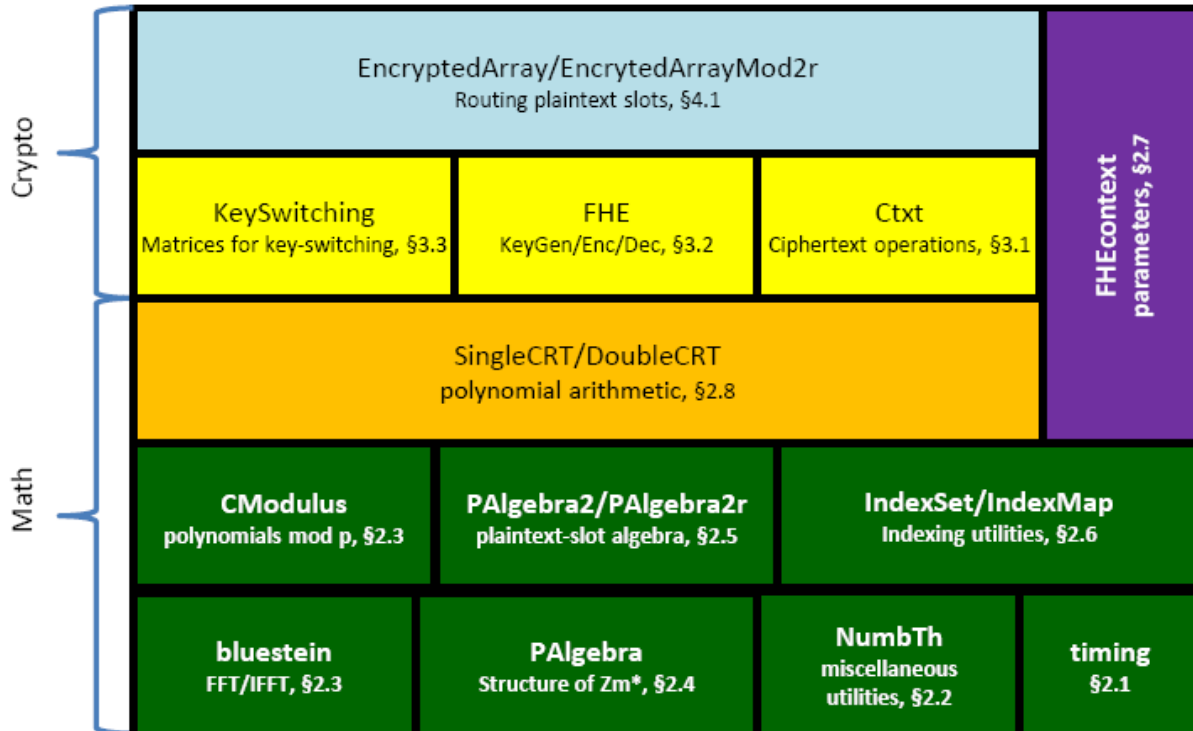


Figure 1: A block diagram of the Homomorphic-Encryption library

4.3 Publications

Full versions of PROCEED AHEAD papers are provided as an attachment to this report as noted in the appendix to this report.

1. “Fully Homomorphic Encryption without Squashing Using Depth-3 Arithmetic Circuits” by Craig Gentry and Shai Halevi[2]

We describe a new approach for constructing fully homomorphic encryption (FHE) schemes. Previous FHE schemes all use the same blueprint from Gentry[1]. First construct a somewhat homomorphic encryption (SWHE) scheme, next "squash" the decryption circuit until it is simple enough to be handled within the homomorphic capacity of the SWHE scheme, and finally "bootstrap" to get a FHE scheme. In all existing schemes, the squashing technique induces an additional assumption: that the sparse subset sum problem (SSSP) is hard.

Our new approach constructs FHE as a hybrid of a SWHE and a multiplicatively homomorphic encryption (MHE) scheme, such as Elgamal. Our construction eliminates the need for the squashing step, and thereby also removes the need to assume the SSSP is hard. We describe a few concrete instantiations of the new method, including a "simple" FHE scheme where we replace SSSP with Decision Diffie-Hellman, an optimization of the simple scheme that let us "compress" the FHE ciphertext into a single Elgamal ciphertext(!), and a scheme whose security can be (quantumly) reduced to the approximate ideal-SIVP.

We stress that the new approach still relies on bootstrapping, but it shows how to bootstrap without having to "squash" the decryption circuit. The main technique is to express the decryption function of SWHE schemes as a depth-3 $(\sum \prod \sum)$ arithmetic circuit of a particular form. When evaluating this circuit homomorphically (as needed for bootstrapping), we temporarily switch to a MHE scheme, such as Elgamal, to handle the \prod part. Due to the special form of the circuit, the switch to the MHE scheme can be done without having to evaluate anything homomorphically. We then translate the result back to the SWHE scheme by homomorphically evaluating the decryption function of the MHE scheme. Using our method, the SWHE scheme only needs to be capable of evaluating the MHE scheme's decryption function, not its own decryption function. We thereby avoid the circularity that necessitated squashing in the original blueprint.

2. "Fully Homomorphic Encryption without Bootstrapping" by Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan[3]

We present a radically new approach to fully homomorphic encryption (FHE) that dramatically improves performance and bases security on weaker assumptions. A central conceptual contribution in our work is a new way of constructing leveled fully homomorphic encryption schemes (capable of evaluating arbitrary polynomial-size circuits), without *Gentry's bootstrapping procedure*.

Specifically, we offer a choice of FHE schemes based on the learning with error (LWE) or ring-LWE (RLWE) problems that have $2^{\tilde{\Omega}}$ security against known attacks. For RLWE, we have:

- A leveled FHE scheme that can evaluate L-level arithmetic circuits with $\tilde{O}(y \cdot L^3)$ per-gate computation— i.e., computation quasi-linear in the security parameter. Security is based on RLWE for an approximation factor exponential in L. This construction does not use the bootstrapping procedure.
- A leveled FHE scheme that uses bootstrapping as an optimization, where the per-gate computation (which includes the bootstrapping procedure) is $\tilde{O}(y^2)$, independent of L. Security is based on the hardness of RLWE for quasi-polynomial factors (as opposed to the sub-exponential factors needed in previous schemes).

We obtain similar results for LWE, but with worse performance. We introduce a number of further optimizations to our schemes. As an example, for circuits of large width – e.g., where a constant fraction of levels have width at least y – we can reduce the per-gate computation of the bootstrapped version to $\tilde{O}(y)$, independent of L, by batching the bootstrapping operation. Previous FHE schemes all required $\Omega(y^{3.5})$ computation per gate.

At the core of our construction is a much more effective approach for managing the noise level of lattice-based ciphertexts as homomorphic operations are performed, using some new techniques recently introduced by Brakerski and Vaikuntanathan[19].

3. "Targeted Malleability: Homomorphic Encryption for Restricted Computations" by Dan Boneh, Gil Segev and Brent Waters[13]

We put forward the notion of targeted malleability: given a homomorphic encryption scheme, in various scenarios we would like to restrict the homomorphic computations one can perform on

encrypted data. We introduce a precise framework, generalizing the foundational notion of non-malleability introduced by Dolev, Dwork, and Naor[20], ensuring that the malleability of a scheme is targeted only at a specific set of "allowable" functions.

In this setting we are mainly interested in the efficiency of such schemes as a function of the number of repeated homomorphic operations. Whereas constructing a scheme whose ciphertext grows linearly with the number of such operations is straightforward, obtaining more realistic (or merely non-trivial) length guarantees is significantly more challenging.

We present two constructions that transform any homomorphic encryption scheme into one that offers targeted malleability. Our constructions rely on standard cryptographic tools and on succinct non-interactive arguments, which are currently known to exist in the standard model based on variants of the knowledge-of-exponent assumption. The two constructions offer somewhat different efficiency guarantees, each of which may be preferable depending on the underlying building blocks.

4. “Trapdoors for Lattices: Simpler, Tighter, Faster, Smaller” by Daniele Micciancio and Chris Peikert[10]

We give new methods for generating and using “strong trapdoors” in cryptographic lattices, which are simultaneously simple, efficient, easy to implement (even in parallel), and asymptotically optimal with very small hidden constants. Our methods involve a new kind of trapdoor, and include specialized algorithms for inverting LWE, randomly sampling SIS preimages, and securely delegating trapdoors. These tasks were previously the main bottleneck for a wide range of cryptographic schemes, and our techniques substantially improve upon the prior ones, both in terms of practical performance and quality of the produced outputs. Moreover, the simple structure of the new trapdoor and associated algorithms can be exposed in applications, leading to further simplifications and efficiency improvements. We exemplify the applicability of our methods with new digital signature schemes and CCA-secure encryption schemes, which have better efficiency and security than the previously known lattice-based constructions.

5. “Homomorphic Evaluation of the AES Circuit” by Craig Gentry, Shai Halevi and Nigel Smart[7]

We describe a working implementation of leveled homomorphic encryption (without bootstrapping) that can evaluate the AES-128 circuit in three different ways. One variant takes under over 36 hours to evaluate an entire AES encryption operation, using NTL (over GMP) as our underlying software platform, and running on a large-memory machine. Using SIMD techniques, we can process over 54 blocks in each evaluation, yielding an amortized rate of just under 40 minutes per block. Another implementation takes just over two and a half days to evaluate the AES operation, but can process 720 blocks in each evaluation, yielding an amortized rate of just over five minutes per block. We also detail a third implementation, which theoretically could yield even better amortized complexity, but in practice turns out to be less competitive.

For our implementations we develop both AES-specific optimizations as well as several “generic” tools for FHE evaluation. These last tools include (among others) a different variant of the Brakerski-Vaikuntanathan key-switching technique that does not require reducing the norm of the ciphertext vector, and a method of implementing the Brakerski-Gentry-Vaikuntanathan modulus-switching transformation on ciphertexts in CRT representation.

6. “Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP” by Zvika Brakerski[6]

We present a new tensoring technique for LWE-based fully homomorphic encryption. While in all previous works, the ciphertext noise grows quadratically ($B \rightarrow B^2 \cdot \text{poly}(n)$) with every multiplication (before “refreshing”), our noise only grows linearly ($B \rightarrow B \cdot \text{poly}(n)$).

We use this technique to construct a scale-invariant fully homomorphic encryption scheme, whose properties only depend on the ratio between the modulus q and the initial noise level B , and not on their absolute values.

Our scheme has a number of advantages over previous candidates: It uses the same modulus throughout the evaluation process (no need for “modulus switching”), and this modulus can take arbitrary form. In addition, security can be classically reduced from the worst-case hardness of the GapSVP problem (with quasi-polynomial approximation factor), whereas previous constructions could only exhibit a quantum reduction from GapSVP.

7. “When Homomorphism Becomes a Liability” by Zvika Brakerski[15]

We show that an encryption scheme cannot have a simple decryption function and be homomorphic at the same time, even with added noise. Specifically, if a scheme can homomorphically evaluate the majority function, then its decryption cannot be weakly-learnable (in particular, linear), even if large decryption error is allowed. (In contrast, without homomorphism, such schemes do exist and are presumed secure, e.g., based on LPN.)

An immediate corollary is that known schemes that are based on the hardness of decoding in the presence of low hamming-weight noise cannot be fully homomorphic. This applies to known schemes such as LPN-based symmetric or public key encryption.

Using these techniques, we show that the recent candidate fully homomorphic encryption, suggested by Bogdanov and Lee (BL)[14], is insecure. In fact, we show two attacks on the BL scheme: One that uses homomorphism, and another that directly attacks a component of the scheme.

8. “Quantum-secure Message Authentication Codes” by Dan Boneh and Mark Zhandry[16]

We construct the first Message Authentication Codes (MACs) that are existentially unforgeable against a quantum chosen message attack. These chosen message attacks model a quantum adversary’s ability to obtain the MAC on a superposition of messages of its choice. We begin by showing that a quantum secure PRF is sufficient for constructing a quantum secure MAC, a fact that is considerably harder to prove than its classical analogue. Next, we show that a variant of Carter-Wegman MACs can be proven to be quantum secure. Unlike the classical settings, we

present an attack showing that a pair-wise independent hash family is insufficient to construct a quantum secure one-time MAC, but we prove that a four-wise independent family is sufficient for one-time security.

9. “Dynamic Proofs of Retrievability via Oblivious RAM” by David Cash and Alptekin Kupcu and Daniel Wichs[21]

Proofs of retrievability allow a client to store her data on a remote server (“in the cloud”) and periodically execute an efficient audit protocol to check that all of the data is being maintained correctly and can be recovered from the server. For efficiency, the computation and communication of the server and client during an audit protocol should be significantly smaller than reading/transmitting the data in its entirety. Although the server is only asked to access a few locations of its storage during an audit, it must maintain full knowledge of all client data to be able to pass.

Starting with the work of Juels and Kaliski[22], all prior solutions to this problem crucially assume that the client data is static and do not allow it to be efficiently updated. Indeed, they all store a redundant encoding of the data on the server, so that the server must delete a large fraction of its storage to ‘lose’ any actual content. Unfortunately, this means that even a single bit modification to the original data will need to modify a large fraction of the server storage, which makes updates highly inefficient. Overcoming this limitation was left as the main open problem by all prior works.

In this work we give the first solution providing proofs of retrievability for dynamic storage, where the client can perform arbitrary reads/writes on any location within her data by running an efficient protocol with the server. At any point in time, the client can execute an efficient audit protocol to ensure that the server maintains the latest version of the client data. The computation and communication complexity of the server and client in our protocols is only polylogarithmic in the size of the client’s data. The starting point of our solution is to split up the data into small blocks and redundantly encode each block of data individually, so that an update inside any data block only affects a few codeword symbols. The main difficulty is to prevent the server from identifying and deleting too many codeword symbols belonging to any single data block. We do so by hiding where the various codeword symbols for any individual data block are stored on the server and when they are being accessed by the client, using the algorithmic techniques of oblivious RAM.

10. “Hardness of SIS and LWE with Small Parameters” by Daniele Micciancio and Chris Peikert[11]

The Short Integer Solution (SIS) and Learning With Errors (LWE) problems are the foundations for countless applications in lattice-based cryptography, and are provably as hard as approximate lattice problems in the worst case. An important question from both a practical and theoretical perspective is how small their parameters can be made, while preserving their worst-case hardness. We prove two main results on SIS and LWE with small parameters. For SIS, we show that the problem retains worst-case hardness for moduli $q \geq \beta \cdot n^\delta$ for any constant $\delta > 0$, where β is the bound on the Euclidean norm of the solution. This improves upon prior results which required $q \geq \beta \cdot \sqrt{n \log n}$, and is essentially optimal since the problem is

trivially easy for $q \leq \beta$. For LWE, we show that it remains hard even when the errors are small (e.g., uniformly random from $\{0, 1\}$), provided that the number of samples is small enough (e.g., linear in the dimension n of the LWE secret). Prior results required the errors to have magnitude at least \sqrt{n} and to come from a Gaussian-like distribution.

11. “How to Delegate Secure Multiparty Computation to the Cloud” by Nishanth Chandran, Rosario Gennaro, Abhishek Jain, Amit Sahai.[23]

We initiate the study of verifiable computation in the presence of many clients who rely on a server to perform computations over inputs privately held by each client. This generalizes the single-client model for verifiable outsourced computation previously studied in the literature. We put forward a computational model and security definitions for this task. We then present a new protocol that allows the clients to securely outsource an arbitrary computation over privately held inputs to a powerful server. At the end the clients will be assured that the result of the computation is correct, while at the same time protecting their data from the server and each other. Our new protocol satisfies the crucial efficiency requirement of outsourced computation where the work of the client is substantially smaller than what is required to compute the function. We use the Gennaro et al. amortized model, whereas the clients are allowed to invest into a one-time computationally expensive preprocessing phase. Additionally our protocol minimizes the interaction between the clients, by requiring only one round of interaction between them for each computation outsourced to the server. Such single round of interaction is necessary if input privacy is to be preserved.

12. “An Equational Approach to Secure Multi-Party Computation” by Daniele Micciancio and Stefano Tessaro[12]

We present a novel framework for the description and analysis of secure computation protocols that is at the same time mathematically rigorous and notationally lightweight and concise. The distinguishing feature of the framework is that it allows to specify (and analyze) protocols in a manner that is largely independent of time, greatly simplifying the study of cryptographic protocols. At the notational level, protocols are described by systems of mathematical equations (over domains), and can be studied through simple algebraic manipulations like substitutions and variable elimination. We exemplify our framework by analyzing in detail two classic protocols: a protocol for secure broadcast, and a verifiable secret sharing protocol, the second of which illustrates the ability of our framework to deal with probabilistic systems, still in a purely equational way.

13. “Semantic Security for the Wiretap Channel” by Mihir Bellare, Stefano Tessaro, and Alexander Vardy [24]

The wiretap channel is a setting where one aims to provide information-theoretic privacy of communicated data based solely on the assumption that the channel from sender to adversary is “noisier” than the channel from sender to receiver. It has developed in the Information and Coding (I&C) community over the last 30 years largely divorced from the parallel development of modern cryptography. This paper aims to bridge the gap with a cryptographic treatment involving advances on two fronts, namely definitions and schemes. On the first front (definitions), we explain that the mis- r definition in current use is weak and propose two

alternatives: mis (based on mutual information) and ss (based on the classical notion of semantic security). We prove them equivalent, thereby connecting two fundamentally different ways of defining privacy and providing a new, strong and well-founded target for constructions. On the second front (schemes), we provide the first explicit scheme with all the following characteristics: it is proven to achieve both security (ss and mis, not just mis-r) and decodability; it has optimal rate; and both the encryption and decryption algorithms are proven to be polynomialtime.

14. “Multi-Instance Security and its Application to Password-Based Cryptography,” by Mihir Bellare, Thomas Ristenpart, and Stephano Tessaro[25]

This paper develops a theory of multi-instance (mi) security and applies it to provide the first proof-based support for the classical practice of salting in password-based cryptography. Mi-security comes into play in settings (like password-based cryptography) where it is computationally feasible to compromise a single instance, and provides a second line of defense, aiming to ensure (in the case of passwords, via salting) that the effort to compromise all of some large number m of instances grows linearly with m . The first challenge is definitions, where we suggest LORX-security as a good metric for mi security of encryption and support this claim by showing it implies other natural metrics, illustrating in the process that even lifting simple results from the si setting to the mi one calls for new techniques. Next we provide a composition-based framework to transfer standard single-instance (si) security to mi-security with the aid of a key-derivation function. Analyzing password-based KDFs from the PKCS#5 standard to show that they meet our indistinguishability-style mi-security definition for KDFs, we are able to conclude with the first proof that per password salts amplify mi-security as hoped in practice. We believe that mi-security is of interest in other domains and that this work provides the foundation for its further theoretical development and practical application.

15. “To Hash or Not to Hash Again? (In)differentiability Results for H^2 and HMAC,” by Yevgeniy Dodis, Thomas Ristenpart, John Steinberger, and Stephano Tessaro[26]

We show that the second iterate $H^2(M) = H(H(M))$ of a random oracle H cannot achieve strong security in the sense of indistinguishability from a random oracle. We do so by proving that indistinguishability for H^2 holds only with poor concrete security by providing a lower bound (via an attack) and a matching upper bound (via a proof requiring new techniques) on the complexity of any successful simulator. We then investigate HMAC when it is used as a general-purpose hash function with arbitrary keys (and not as a MAC or PRF with uniform, secret keys). We uncover that HMAC’s handling of keys gives rise to two types of weak key pairs. The first allows trivial attacks against its indistinguishability; the second gives rise to structural issues similar to that which ruled out strong indistinguishability bounds in the case of H^2 . However, such weak key pairs do not arise, as far as we know, in any deployed applications of HMAC. For example, using keys of any fixed length shorter than $d - 1$, where d is the block length in bits of the underlying hash function, completely avoids weak key pairs. We therefore conclude with a positive result: a proof that HMAC is indistinguishable from a RO (with standard, good bounds) when applications use keys of a fixed length less than $d - 1$.

16. “Design and Implementation of a Homomorphic-Encryption Library”, by Shai Halevi and

Victor Shoup.

We describe the design and implementation of a software library that implements the Brakerski-Gentry-Vaikuntanathan (BGV) homomorphic encryption scheme, along with many optimizations to make homomorphic evaluation run faster, focusing mostly on effective use of the Smart-Vercauteren ciphertext packing techniques. Our library is written in C++ and uses the NTL mathematical library.

17. “Using Homomorphic Encryption for large Scale Statistical Analysis”, by David Wu, Jacob Haven and Dan Boneh.

We describe in a Viewgraph format, the scale-invariant leveled fully homomorphic encryption scheme. With this scheme we are able to use batching and CRT-based message encoding to perform large scale statistical analysis on millions of data points and data of moderate dimension. The chart progresses through; the motivation, the approach, the theory of computation on large integers, the Client side and Server sides of the Homomorphic Encryption Schemes, Experimental Timing Results and Conclusions.

5.0 Conclusions and Recommendations

The PROCEED AHEAD team has delivered outstanding results in just the first half (two years) of the intended four year project. Our results spanned from both fundamental theoretical contributions to the development of Fully Homomorphic Encryption (FHE), special forms of FHE, better lattice design, through applications such as delegation of computations and understanding of the underlyings of multiparty computations, to implementations and optimizations of FHE.

There lies significant opportunity to build upon our work from the first two years of the PROCEED AHEAD project including additional aspects of computation delegation. There is also an opening to apply our prototype system for performing statistical analysis on large scale data computing the mean and covariance of multivariate data and performing linear regression over encrypted datasets. Also, the Homomorphic Encryption (HE) software library[18] requires the cooperation from PROCEED program partners and the cryptographic research community at large to advance it from its current "proof of concept" state into a fully homomorphic encryption scheme applicable to real world tasks such as network guards and other computations on encrypted data such as curve fitting (e.g., least-squares fit).

Even with the major advances in the state of HE over the last few years, both the size of HE ciphertext and the complexity of computing them remain quite high. An obvious direction for future work is to find additional optimizations to reduce this overhead further. One direction which was not explored but seems to have large potential, is finding a cheaper method to replace Gentry's bootstrapping technique. Namely, it is still plausible that we can reduce the noise in the ciphertext by applying a cheaper transformation than full homomorphic decryption.

The team has worked intensely on the AHEAD project, collaborating with other participants within the PROCEED program, and has delivered results in a pace that even surprised us! IBM has enjoyed working on the project and will take leave (resulting from limitation of funds) while Stanford and UCSD continue to work within the PROCEED program.

6.0 References

- [1] Gentry, Craig, "A fully homomorphic encryption scheme" *PhD thesis*, Stanford University, 2009. <http://crypto.stanford.edu/craig>.
- [2] Gentry, Craig, and Halevi, Shai, "Fully Homomorphic Encryption without Squashing Using Depth-3 Arithmetic Circuits," *FOCS* pps 107-109, 2011, <http://eprint.iacr.org/2011/279>
- [3] Brakerski, Zvika; Gentry, Craig; and Vaikuntanathan, Vinod, "Fully Homomorphic Encryption without Bootstrapping," *ITCS*, pps 309-325, 2012, <http://eprint.iacr.org/2011/277>
- [4] Gentry, Craig; Halevi, Shai; and Smart, Nigel, "Fully Homomorphic Encryption with Polylog Overhead," *Eurocrypt* pps 465-482, 2012, <http://eprint.iacr.org/2011/566>
- [5] Gentry, Craig; Halevi, Shai; and Smart, Nigel, "Better Bootstrapping in Fully Homomorphic Encryption," *PKC* 2012. **LNCS** vol. 7293, pps 1-16, <http://eprint.iacr.org/2011/680>
- [6] Brakerski, Zvika, "Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP," *Crypto*, pps 868-886, 2012, <http://eprint.iacr.org/2012/078>
- [7] Gentry, Craig; Halevi, Shai; and Smart, Nigel, "Homomorphic Evaluation of the AES Circuit," *CRYPTO* pps 850-867, 2012, <http://eprint.iacr.org/2012/099>
- [8] Brakerski, Zvika; Gentry, Craig; and Halevi, Shai, "Packed Ciphertexts in LWE-based Homomorphic Encryption," *PKC* 2012, <http://eprint.iacr.org/2012/565>
- [9] Gentry, Craig, and Halevi, Shai, "Implementing Gentry's Fully-Homomorphic Encryption Scheme," *EUROCRYPT* **LNCS** vol. **6632**, 2 pps 129-148, 011, <http://eprint.iacr.org/2010/520>
- [10] Micciancio, Daniele, and Peikert, Chris, "Trapdoors for Lattices: Simpler, Tighter, Faster, Smaller," *Eurocrypt* pps 700-718, 2012, http://link.springer.com/chapter/10.1007%2F978-3-642-29011-4_41
- [11] Micciancio, Daniele, and Peikert, Chris, "Hardness of SIS and LWE with Small Parameters," submitted to *CCC* 2013. <http://eprint.iacr.org/2013/069.pdf>
- [12] Micciancio, Daniele, and Tessaro, Stefano, "An Equational Approach to Secure Multi-Party Computation," *ITCS* pps 355-372, 2013, <http://cseweb.ucsd.edu/~daniele/papers/ITCS13.pdf>
- [13] Boneh, Dan; Segev, Gil; and Waters, Brent, "Targeted Malleability: Homomorphic Encryption for Restricted Computations", *ITCS*, pp 350-366, 2012, <http://eprint.iacr.org/2011/311>

- [14] Bogdanov, Andrej, and Lee, Chin Ho, "Homomorphic encryption from codes" *Cryptology ePrint Archive*, **Report 2011/622**, 2011. <http://eprint.iacr.org/2011/622>
- [15] Brakerski, Zvika, "When Homomorphism Becomes a Liability," *TCC* 2013, <http://eprint.iacr.org/2012/225>
- [16] Boneh, Dan, and Zhandry, Mark, "Quantum-secure Message Authentication Codes," *Eurocrypt*, 2013, <http://eprint.iacr.org/2012/606>
- [17] Wu, David; Haven, Jacob; and Boneh, Dan, "Using Homomorphic Encryption for Large Scale Statistical Analysis," *CURIS* 2012. www.stanford.edu/~dwu4/CURISPoster.pdf
- [18] Halevi, Shai, and Shoup, Victor, "Design and Implementation of a Homomorphic-Encryption Library," to be submitted. <http://researcher.ibm.com/researcher/files/us-shaih/he-library.pdf>
- [19] Brakerski, Zvika, and Vaikuntanathan, Vinod, "Efficient fully homomorphic encryption from (standard) LWE" *FOCS* 2011, <http://eprint.iacr.org/2011/344>
- [20] Dolev, D.; Dwork, C.; and Naor, M., "Non-malleable cryptography" *SIAM Journal on Computing*, **30(2)**, pps 391–437, 2000, <http://noodle.cs.huji.ac.il/~dolev/pubs/nmc.pdf>
- [21] Cash, David; Kupcu, Alptekin; and Wichs, Daniel, "Dynamic Proofs of Retrievability via Oblivious RAM," *Eurocrypt* 2013, <http://eprint.iacr.org/2012/550>
- [22] Juels, A. and Kaliski, B., "PORs: Proofs of Retrievability for Large Files," *ACM CCS*, pp. 584—597. 2007
- [23] Chandran, Nishanth; Gennaro, Rosario; Jain, Abhishek; and Sahai, Amit, "How to Delegate Secure Multiparty Computation to the Cloud," to be submitted.
- [24] Bellare, Mihir; Tessaro, Stephano; and Vardy, Alexander, "Semantic Security for the Wiretap Channel," *CRYPTO* 2012, <http://cseweb.ucsd.edu/~mihir/papers/wiretap-crypto12.pdf>
- [25] Bellare, Mihir; Ristenpart, Thomas; and Tessaro, Stephano, "Multi-Instance Security and its Application to Password-Based Cryptography," *CRYPTO* 2012, <http://eprint.iacr.org/2012/196.pdf>
- [26] Dodis, Yevgeniy; Ristenpart, Thomas; Steinberger, John, and Tessaro, Stephano, "To Hash or Not to Hash Again? (In)differentiability Results for H2 and HMAC," *CRYPTO* 2012, <http://people.csail.mit.edu/tessaro/papers/h2fullprelim.pdf>

7.0 Appendix

Please see Attachment 1 – Publications (PROCEED AHEAD Final Report Full Papers Attachment 1 Mar 2013.pdf) for the following publications:

1. Fully Homomorphic Encryption without Squashing Using Depth-3 Arithmetic Circuits
2. Fully Homomorphic Encryption without Bootstrapping
3. Targeted Malleability: Homomorphic Encryption for Restricted Computations
4. Trapdoors for Lattices: Simpler, Tighter, Faster, Smaller
5. Homomorphic Evaluation of the AES Circuit
6. Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP
7. When Homomorphism Becomes a Liability
8. Quantum-Secure Message Authentication Codes
9. Dynamic Proofs of Retrievability via Oblivious RAM
10. Hardness of SIS and LWE with Small Parameters
11. How to Delegate Secure Multiparty Computation to the Cloud
12. An Equational Approach to Secure Multi-party Computation
13. Semantic Security for the Wiretap Channel
14. Multi-Instance Security and its Application to Password-Based Cryptography
15. To Hash or Not to Hash Again (In)differentiability Results for H^2 and HMAC
16. FHE Library - Design and Implementation of a Homomorphic-Encryption Library
17. Using Homomorphic Encryption for Large Scale Statistical Analysis

8.0 List of Symbols, Abbreviations, and Acronyms

Abbreviation	Definition
AES	Advanced Encryption Standard
AHEAD	Advancing Homomorphic Encryption its Applications and Derivatives
API	Application Programmers Interface
BGV	Brakerski-Gentry-Vaikuntanathan
BL	Bogdanov and Lee
CCA	Chosen Cyphertext Attack
CRT	Chinese Remainder Theorem
DARPA	Defense Advanced Research Projects Agency
FHE	Fully Homomorphic Encryption
GapSVP	GapShort Vector Problem
GMP	The GNU MP Bignum Library
HE	Homomorphic Encryption
IBM	International Business Machines Corporation
LPN	Learning Parity with Noise
LWE	Learning With Errors
MAC	Message Authentication Code
MHE	Multiplicatively Homomorphic Encryption
NTL	Number Theory Library
PI	Principal Investigator
PRF	Pseudo Random Function
PROCEED	PROgramming Computation on EncryptEd Data
RAM	Random Access Memory
RLWE	Ring Learning With Errors
SIMD	Single Instruction Multiple Data
SIS	Short Integer Solution
SSSP	Sparse Subset Sum Problem
SWHE	SomeWhat Homomorphic Encryption
TA	Technical Area
UCSD	University of California, San Diego

9.0 Public Affairs Approval of papers in Appendix

1. Fully Homomorphic Encryption without Squashing Using Depth-3 Arithmetic Circuits.
Approved for Public Release, DISTAR Case # 18107, October 25, 2011

2. Fully Homomorphic Encryption without Bootstrapping.
Approved for Public Release, DISTAR Case #17837, August 8, 2011

3. Targeted Malleability: Homomorphic Encryption for Restricted Computations.
Approved for Public Release, Contracted Fundamental Research

4. Trapdoors for Lattices: Simpler, Tighter, Faster, Smaller.
Approved for Public Release, Contracted Fundamental Research

5. Homomorphic Evaluation of the AES Circuit.
Approved for Public Release, DISTAR Case # 19368, June 11, 2012

6. Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP.
Approved for Public Release, Contracted Fundamental Research

7. When Homomorphism Becomes a Liability.
Approved for Public Release, Contracted Fundamental Research

8. Quantum-Secure Message Authentication Codes.
Approved for Public Release, Contracted Fundamental Research

9. Dynamic Proofs of Retreivability via Oblivious RAM.
Approved for Public Release, DISTAR Case # 19972, October 1, 2012

10. Hardness of SIS and LWE with Small Parameters.
Approved for Public Release, Contracted Fundamental Research

11. How to Delegate Secure Multiparty Computation to the Cloud.
Approved for Public Release, Contracted Fundamental Research

12. An Equational Approach to Secure Multi-Party Computation.
Approved for Public Release, Contracted Fundamental Research

13. Semantic Security for the Wiretap Channel.
Approved for Public Release, Contracted Fundamental Research

Public Affairs Approvals for papers in Appendix, Continued.

14. Multi-Instance Security and its Application to Password-Based Cryptography.

Approved for Public Release, Contracted Fundamental Research

15. To Hash or Not to Hash Again? (In)differentiability Results for H^2 and HMAC.

Approved for Public Release, Contracted Fundamental Research

16. FHE Library - Design and Implementation of a Homomorphic-Encryption Library

Approved for Public Release, DISTAR Case # 20493, January 18, 2013

17. Using Homomorphic Encryption for Large Scale Statistical Analysis

Approved for Public Release, Contracted Fundamental Research

Fully Homomorphic Encryption without Squashing Using Depth-3 Arithmetic Circuits

Craig Gentry and Shai Halevi
IBM T.J. Watson Research Center

September 14, 2011

Abstract

We describe a new approach for constructing fully homomorphic encryption (FHE) schemes. Previous FHE schemes all use the same blueprint from [Gentry 2009]: First construct a somewhat homomorphic encryption (SWHE) scheme, next “squash” the decryption circuit until it is simple enough to be handled within the homomorphic capacity of the SWHE scheme, and finally “bootstrap” to get a FHE scheme. In all existing schemes, the squashing technique induces an additional assumption: that the sparse subset sum problem (SSSP) is hard.

Our new approach constructs FHE as a hybrid of a SWHE and a multiplicatively homomorphic encryption (MHE) scheme, such as Elgamal. Our construction eliminates the need for the squashing step, and thereby also removes the need to assume the SSSP is hard. We describe a few concrete instantiations of the new method, including a “simple” FHE scheme where we replace SSSP with Decision Diffie-Hellman, an optimization of the simple scheme that let us “compress” the FHE ciphertext into a single Elgamal ciphertext(!), and a scheme whose security can be (quantumly) reduced to the approximate ideal-SIVP.

We stress that the new approach still relies on bootstrapping, but it shows how to bootstrap without having to “squash” the decryption circuit. The main technique is to express the decryption function of SWHE schemes as a depth-3 ($\sum \prod \sum$) arithmetic circuit of a particular form. When evaluating this circuit homomorphically (as needed for bootstrapping), we temporarily switch to a MHE scheme, such as Elgamal, to handle the \prod part. Due to the special form of the circuit, the switch to the MHE scheme can be done without having to evaluate anything homomorphically. We then translate the result back to the SWHE scheme by homomorphically evaluating the decryption function of the MHE scheme. Using our method, the SWHE scheme only needs to be capable of evaluating the MHE scheme’s decryption function, not its own decryption function. We thereby avoid the circularity that necessitated squashing in the original blueprint.

Key words. Arithmetic Circuits, Depth-3 Circuits, Homomorphic Encryption, Symmetric Polynomials

Contents

1	Introduction	1
1.1	Our Main Technical Innovation	1
1.2	An Illustration of an Elgamal-Based Instantiation	2
1.3	Leveled FHE Based on Worst-Case Hardness	3
2	Decryption as a Depth-3 Arithmetic Circuit	4
2.1	Restricted Depth-3 Arithmetic Circuits	4
2.2	Lattice-Based Somewhat-Homomorphic Cryptosystems	6
2.3	Decryption Using a Restricted Depth-3 Circuit	6
3	Leveled FHE from SWHE and MHE	7
3.1	Notations	7
3.2	Compatible SWHE and MHE Schemes	8
3.3	Chimeric Leveled FHE: The Construction	9
4	Optimizations	10
4.1	Computing Only One Product	10
4.2	Short FHE Ciphertexts: Decryption as a Pure Symmetric Polynomial	11
	References	12
A	Instantiations of Chimeric FHE	14
A.1	The Homomorphic Capacity of SWHE Schemes	14
A.2	Elgamal-based Instantiation	14
A.3	Leveled FHE Based on Worst-Case Hardness	16
A.3.1	Decryption under S_{ml}	17
A.3.2	The SWHE scheme L_{rg}	18
A.3.3	Setting the parameters.	18
B	Proof of Lemma 1	19

1 Introduction

Fully homomorphic encryption allows anyone to perform arbitrarily computations on encrypted data, despite not having the secret decryption key. Several fully homomorphic encryption (FHE) schemes appeared recently [Gen09b, vDGHV10, SV10, GH11], all following the same blueprint as Gentry’s original construction [Gen09b, Gen09a]:

- 1. SWHE.** Construct a somewhat homomorphic encryption (SWHE) scheme – roughly, a scheme that can evaluate low-degree polynomials homomorphically.
- 2. Squash.** “Squash” the decryption function of the SWHE scheme, until decryption can be expressed as polynomial of degree low enough to be handled within the homomorphic capacity of the SWHE scheme, with enough capacity left over to evaluate a NAND gate. This is done by adding a “hint” to the public key – namely, a large set of elements that has a secret sparse subset that sums to the original secret key.
- 3. Bootstrap.** Given a SWHE scheme that can evaluate its decryption function (plus a NAND), apply Gentry’s transformation to get a “leveled”¹ FHE scheme.

In this work we construct leveled FHE by combining a SWHE scheme with a “compatible” multiplicatively homomorphic encryption (MHE) scheme (such as Elgamal) in a surprising way. Our construction still relies on bootstrapping, but it does not use squashing and does not rely on the assumed hardness of the sparse subset sum problem (SSSP). Using the new method, we construct a “simple” leveled FHE scheme where SSSP is replaced with Decision Diffie-Hellman. We also describe an optimization of this simple scheme where at one point during the bootstrapping process, the entire leveled FHE ciphertext consists of a single MHE (e.g., Elgamal) ciphertext! Finally, we show that it is possible to replace the MHE scheme by an additively homomorphic encryption (AHE) scheme that encrypts discrete logarithms. This allows us to construct a leveled FHE scheme whose security is based *entirely* on the worst-case hardness of the shortest independent vector problem over ideal lattices (ideal-SIVP) (compare [Gen10]). As in Gentry’s original blueprint, we obtain a pure FHE scheme by assuming circular security. At present, our new approach does not improve efficiency, aside from the optimization that reduces the ciphertext length.

1.1 Our Main Technical Innovation

Our main technical innovation is a new way to evaluate homomorphically the decryption circuits of the underlying SWHE schemes. Decryption in these schemes involves computing a threshold function, that can be expressed as a multilinear symmetric polynomial. Previous works [Gen09b, vDGHV10, SV10, GH11] evaluated those polynomials in the “obvious way” using *boolean* circuits. Instead, here we use Ben-Or’s observation (reported in [NW97]) that multilinear symmetric polynomials can be computed by depth-3 ($\sum \prod \sum$) *arithmetic* circuits over \mathbb{Z}_p for large enough prime p . Let $e_k(\cdot)$ be the n -variable degree- k elementary symmetric polynomial, and consider a vector $\vec{x} = \langle x_1, \dots, x_n \rangle \in \{0, 1\}^n$. The value of $e_k(\vec{x})$ is simply the coefficient of z^{n-k} in the univariate polynomial $P(z) = \prod_{i=1}^n (z + x_i)$. This coefficient can be computed by fixing an arbitrary set $A = \{a_1, \dots, a_{n+1}\} \subseteq \mathbb{Z}_p$, then evaluating the polynomial $P(z)$ at the points in A to obtain

¹In a “leveled” FHE scheme, the size of the public key is linear in the depth of the circuits to evaluate. A “pure” FHE scheme (with a fixed-sized public key) can be obtained by assuming “circular security” – namely, that it is safe to encrypt the leveled FHE secret key under its own public key.

$t_j = P(a_j)$, and finally interpolating the coefficient of interest as a linear combination of the t_j 's. The resulting circuit has the form

$$e_k(\vec{x}) = \sum_{j=1}^{n+1} \lambda_{jk} \prod_{i=1}^n (a_j + x_i) \pmod{p}, \quad (1)$$

where λ_{jk} 's are the interpolation coefficients, which are some known constants in \mathbb{Z}_p . Any multilinear symmetric polynomial over \vec{x} can be computed as a linear combination of the $e_k(\vec{x})$'s, and thus has the same form (with different λ 's).

By itself, Ben-Or's observation is not helpful to us, since (until now) we did not know how to bootstrap unless the polynomial *degree* of the decryption function is low. Ben-Or's observation does not help us lower the degree (it actually *increases* the degree).² Our insight is that we can evaluate the \prod part by temporarily working with a MHE scheme, such as Elgamal [ELG85]. We first use a simple trick to get an encryption under the MHE scheme of all the $(a_j + x_i)$ terms in Ben-Or's circuit, then use the multiplicative homomorphism to multiply them, and finally convert them back to SWHE ciphertexts to do the final sum. Conversion back from MHE to SWHE is done by running the MHE scheme's decryption circuit homomorphically within the SWHE scheme, which may be expensive. However, the key point is that the degree of the translation depends only on the MHE scheme and *not on the SWHE scheme*. This breaks the self-referential requirement of being able to evaluate its *own* decryption circuit, hence obviating the need for the squashing step. Instead, we can now just increase the parameters of the SWHE scheme until it can handle the MHE scheme's decryption circuit.

1.2 An Illustration of an Elgamal-Based Instantiation

Perhaps the simplest illustration of our idea is using Elgamal encryption to do the multiplication. Let $p = 2q + 1$ be a safe prime. Elgamal messages and ciphertext components will live in $\text{QR}(p)$, the group of quadratic residues modulo p . We also use a SWHE scheme with plaintext space \mathbb{Z}_p . (All previous SWHE schemes can be adapted to handle this large plaintext space). We also require the SWHE scheme to have a "simple" decryption function that can be expressed as a "restricted" depth-3 arithmetic circuit. These terms are defined later in Section 2, for now we just mention that all known SWHE schemes [Gen09b, vDGHV10, SV10, GH11] meet this condition

For simplicity of presentation here, imagine that the SWHE secret key is a bit vector $\vec{s} = (s_1, \dots, s_n) \in \{0, 1\}^n$, the ciphertext that we want to decrypt is also a bit vector $\vec{c} = (c_1, \dots, c_n) \in \{0, 1\}^n$, and that decryption works by first computing $x_i \leftarrow s_i \cdot c_i$ for all i , and then running the $\sum \prod \sum$ circuit, taking \vec{x} as input. Imagine that decryption simply performs something like interpolation – namely, it computes $f(\vec{x}) = \sum_{j=1}^{n+1} \lambda_j \prod_{i=1}^n (a_j + x_i)$, where the a_j 's and λ_j 's are publicly known constants in \mathbb{Z}_p .

To enable bootstrapping, we provide (in the public key) the Elgamal secret key encrypted under the SWHE public key, namely we encrypt the bits of the secret Elgamal exponent e individually under the SWHE scheme. We also use a special form of encryption of the SWHE secret key under the Elgamal public key. Namely, for each secret-key bit s_i and each public constant a_j , we provide

²The degree of $P(z)$ is n , whereas in the previous blueprint Gentry's squashing technique is used to ensure that the Hamming weight of \vec{x} is at most $m \ll n$, so that it suffices to compute $e_k(\vec{x})$ only for $k \leq m$.

an ElGamal encryption of the value $a_j + s_i \in \mathbb{Z}_p$. The public values a_j 's are chosen so that both $a_j, a_j + 1 \in \text{QR}(p)$, so that $a_j + s_i$ is always in the ElGamal plaintext space.³

Now let $\vec{c} \in \{0, 1\}^n$ be a SWHE ciphertext that we want to decrypt *homomorphically*. First, for each (i, j) , we obtain an ElGamal ciphertext that encrypts $a_j + (s_i \cdot c_i)$ as follows: if $c_i = 0$ then $a_j + (s_i \cdot c_i) = a_j$, so we simply generate a fresh encryption of the public value a_j . On the other hand, if $c_i = 1$ then $a_j + (s_i \cdot c_i) = a_j + s_i$, so we use the encryption of $a_j + s_i$ from the public key. (Note how the “restricted” form of these sums $a_j + x_i$ makes it possible to put in the public key all the ElGamal ciphertexts that are needed for these sums.)

Next we use ElGamal’s multiplicative homomorphism for the \prod part of the circuit, thus getting ElGamal encryptions of the values $\lambda_j \cdot P(a_j)$ (where $P(z) = \prod_i (z + x_i)$). We then convert these ElGamal encryptions into SWHE encryptions of the same values in \mathbb{Z}_p by homomorphically evaluating the ElGamal decryption, using the SWHE encryption of the ElGamal secret exponent from the public key. Denote by e_i the i ’th bit of the secret exponent e (so the public key includes an SWHE encryption of e_i), and let $(y, z) = (g^r, m \cdot g^{-er})$ be an ElGamal ciphertext to be converted. We compute $y^{2^i} - 1 \bmod p$ for all i , then compute SWHE ciphertexts that encrypt the powers

$$y^{e_i \cdot 2^i} = e_i y^{2^i} + (1 - e_i) y^0 = e_i (y^{2^i} - 1) + 1,$$

and then use multiplicative homomorphism of the SWHE scheme to multiply these powers and obtain an encryption of y^e . (This requires degree $\lceil \log q \rceil$). Finally, inside the SWHE scheme, we multiply the encryption of y^e by the known value z , thereby obtaining a SWHE ciphertext that encrypts m .

At this point, we have SWHE ciphertexts that encrypt the results of the \prod operations – the values $\lambda_j \cdot P(a_j)$. We now use the SWHE scheme’s additive homomorphism to finish off the depth-3 circuit, thus completing the homomorphic decryption. We can now compute another MULT or ADD operation, before running homomorphic decryption again to “refresh” the result, ad infinitum.

As explained above, using this approach the SWHE scheme only needs to evaluate polynomials that are slightly more complex than the MHE scheme’s decryption circuit. Specifically, for ElGamal we need to evaluate polynomials of degree $2 \lceil \log q \rceil$. We can use any of the prior SWHE schemes from the literature, and set the parameters large enough to handle these polynomials. The security of the resulting leveled FHE scheme is based on the security of its component SWHE and MHE schemes.

We also show that by a careful choice of the constants a_j , we can set things up so that we always have $P(a_j) = w_j \cdot P(a_1)^{e_j}$ for some known constants $e_j, w_j \in \mathbb{Z}_p$. Hence we can compute all the ElGamal ciphertexts at the output of the Π layer given just the ElGamal ciphertext that encrypts $P(a_1)$, which yields a compact representation of the ciphertext.

1.3 Leveled FHE Based on Worst-Case Hardness

We use similar ideas to get a leveled FHE scheme whose security is based entirely on the (quantum) worst-case hardness of ideal-SIVP. At first glance this may seem surprising: how can we use a lattice-based scheme as our MHE scheme when current lattice-based schemes do not handle multiplication very well? (This was the entire reason the old blueprint required squashing!) We get around this

³An amusing exercise: Prove that the number of a_j ’s with $a_j, a_j + 1 \in \text{QR}(p)$ is $(p - 3)/4$ when $p = 3 \bmod 4$ and $(p - 5)/4$ when $p = 1 \bmod 4$. See Lemma 5 for the answer.

apparent problem by replacing the MHE scheme with an *additively* homomorphic encryption (AHE) scheme, applied to *discrete logs*.

In more detail, as in the Elgamal-based instantiation, the SWHE scheme uses plaintext space \mathbb{Z}_p for prime $p = 2q + 1$. But p is chosen to be a *small prime*, polynomial in the security parameter, so it is easy to compute discrete logs modulo p . The plaintext space of the AHE scheme is \mathbb{Z}_q , corresponding to the space of exponents of a generator g of \mathbb{Z}_p^* . Rather than encrypting in the public key the values $a_j + s_i$ (as in the Elgamal instantiation), we provide AHE ciphertexts that encrypt the values $\text{DL}_g(a_j + s_i) \in \mathbb{Z}_q$, and use the same trick as above to get AHE ciphertexts that encrypt the values $\text{DL}_g(a_j + (s_i \cdot c_i))$. We homomorphically add these values, getting an AHE encryption of $\text{DL}_g(\lambda_j \cdot P(a_j))$. Finally, we use the SWHE scheme to homomorphically compute the AHE decryption followed by exponentiation, getting SWHE encryption of the values $\lambda_j \cdot P(a_j)$, which we add within the SWHE scheme to complete the bootstrapping.

As before, the SWHE scheme only needs to support the AHE decryption (and exponentiation modulo the small prime p), thus we don't have the self-reference problem that requires squashing. We note, however, that lattice-based additively-homomorphic schemes are not completely error free, so one must set the parameters so that it supports sufficient number of summands. Since the dependence of the AHE noise on the number of summands is very weak (only logarithmic), this can be done without the need for squashing. See Section A.3 for more details on this construction.

2 Decryption as a Depth-3 Arithmetic Circuit

Recall that, in Gentry's FHE, we “refresh” a ciphertext c by expressing decryption of this ciphertext as a function $D_c(s)$ in the secret key s , and evaluating that function homomorphically. Below, we describe “restricted” depth-3 circuits, sketch a “generic” lattice based construction that encompasses known SWHE schemes (up to minor modifications), and show how to express its decryption function $D_c(s)$ as a restricted depth-3 circuit over a large enough ring \mathbb{Z}_p . We note that Klivans and Sherstov [KS06] have already shown that the decryption functions of Regev's cryptosystems [Reg04, Reg09] can be computed using depth-3 circuits.

2.1 Restricted Depth-3 Arithmetic Circuits

In our construction, the circuit that computes $D_c(s)$ depends on the ciphertext c only in a very restricted manner. By “restricted” we roughly mean that the bottom sums in the depth-3 circuit must come from a fixed (polynomial-size) set \mathcal{L} of polynomials, where \mathcal{L} itself is independent of the ciphertext. Thus, the bottom sums used in the circuit can depend on the ciphertext only to the extent that the ciphertext is used to *select* which and how many of the polynomials in \mathcal{L} are used as bottom sums in the circuit.

Definition 1 (Restricted Depth-3 Circuit). *Let $\mathcal{L} = \{L_j(x_1, \dots, x_n)\}$ be a set of polynomials, all in the same n variables. An arithmetic circuit C is an \mathcal{L} -restricted depth-3 circuit over (x_1, \dots, x_n) if there exists multisets $S_1, \dots, S_t \subseteq \mathcal{L}$ and constants $\lambda_0, \lambda_1, \dots, \lambda_t$ such that*

$$C(\vec{x}) = \lambda_0 + \sum_{i=1}^t \lambda_i \cdot \prod_{L_j \in S_i} L_j(x_1, \dots, x_n),$$

The degree of C with respect to \mathcal{L} is $d = \max_i |S_i|$ (we also call it the \mathcal{L} -degree of C).

Remark 1. In all our instantiations of decryption circuits for known SWHE schemes, the L_j 's happen to be linear. However, our generic construction in Section 3 does not require that they be linear (or even low degree).

To express decryption as restricted circuit as above, we use Ben-Or's observation that multilinear symmetric polynomials can be computed by restricted depth-3 arithmetic circuits that perform interpolation. Recall that a *multilinear symmetric polynomial* $M(\vec{x})$ is a symmetric polynomial where, for each i , every monomial is of degree at most 1 in x_i ; there are no high powers of x_i . A simple fact is that every multilinear symmetric polynomial $M(\vec{x})$ is a linear combination of the elementary symmetric polynomials: $M(\vec{x}) = \sum_{i=0}^n \ell_i \cdot e_i(\vec{x})$, where $e_i(\vec{x})$ is the sum of all degree- i monomials that are the product of i distinct variables. Also, for every symmetric polynomial $S(\vec{x})$, there is a multilinear symmetric polynomial $M(\vec{x})$ that agrees with $S(\vec{x})$ on all binary vectors $\vec{x} \in \{0, 1\}$. The reason is that $x_i^k = x_i$ for $x_i \in \{0, 1\}$, and therefore all higher powers in $S(\vec{x})$ can be "flattened"; the end result is multilinear symmetric. The following lemma states Ben-Or's observation formally.

Lemma 1 (Ben-Or, reported in [NW97]). *Let $p \geq n + 1$ be a prime, let $A \subseteq \mathbb{Z}_p$ have cardinality $n + 1$, let $\vec{x} = (x_1, \dots, x_n)$ be variables, and denote $\mathcal{L}_A \stackrel{\text{def}}{=} \{(a + x_i) : a \in A, 1 \leq i \leq n\}$. For every multilinear symmetric polynomial $M(\vec{x})$ over \mathbb{Z}_p , there is a circuit $C(\vec{x})$ such that:*

- *C is a \mathcal{L}_A -restricted depth-3 circuit over \mathbb{Z}_p such that $C(\vec{x}) \equiv M(\vec{x})$ (in \mathbb{Z}_p).*
- *C has $n + 1$ product gates of \mathcal{L}_A -degree n , one gate for each value $a_j \in A$, with the j 'th gate computing the value $\lambda_j \cdot P(a_j) = \prod_i (a_j + x_i)$ for some scalar λ_j .*
- *A description of C can be computed efficiently given the values $M(\vec{x})$ at all $\vec{x} = 1^i 0^{n-i}$.*

The final bullet clarifies that Ben-Or's observation is constructive – we can compute the restricted depth-3 representation from any initial representation that lets us evaluate M . For completeness, we prove Lemma 1 in Appendix B.

In some cases, it is easier to work with univariate polynomials. The following fact, captured in Lemma 2, will be useful for us: Suppose $f(x)$ is an arbitrary univariate function and we want to compute $f(\sum b_i \cdot t_i)$, where the b_i 's are bits and the t_i 's are small (polynomial). Then, we can actually express this computation as a multilinear symmetric polynomial, and hence a restricted depth-3 circuit in the b_i 's.

Lemma 2. *Let T, n be positive integers, and $f(x)$ a univariate polynomial over \mathbb{Z}_p (for p prime, $p \geq Tn + 1$). Then there is a multilinear symmetric polynomial $M_f(\cdot)$ on Tn variables such that for all $t_1, \dots, t_n \in \{0, \dots, T\}$,*

$$f(b_1 \cdot t_1 + \dots + b_n \cdot t_n) = M_f(\underbrace{b_1, \dots, b_1}_{t_1 \text{ times}}, \underbrace{0, \dots, 0}_{T-t_1 \text{ times}}, \underbrace{b_2, \dots, b_2}_{t_2 \text{ times}}, \underbrace{0, \dots, 0}_{T-t_2 \text{ times}}, \dots, \underbrace{b_n, \dots, b_n}_{t_n \text{ times}}, \underbrace{0, \dots, 0}_{T-t_n \text{ times}})$$

for all $\vec{b} \in \{0, 1\}^n$. Moreover, a representation of M_f as a \mathcal{L}_A -restricted depth-3 circuit can be computed in time $\text{poly}(Tn)$ given oracle access to f .

Proof. Define a Tn -variate polynomial $g : \mathbb{Z}_p^{Tn} \rightarrow \mathbb{Z}_p$ as $g(\vec{x}) = f(\sum x_i)$, then g is symmetric and we have

$$f(b_1 \cdot t_1 + \dots + b_n \cdot t_n) = g(\underbrace{b_1, \dots, b_1}_{t_1 \text{ times}}, \underbrace{0, \dots, 0}_{T-t_1 \text{ times}}, \underbrace{b_2, \dots, b_2}_{t_2 \text{ times}}, \underbrace{0, \dots, 0}_{T-t_2 \text{ times}}, \dots, \underbrace{b_n, \dots, b_n}_{t_n \text{ times}}, \underbrace{0, \dots, 0}_{T-t_n \text{ times}}).$$

As noted above, there is a multilinear symmetric polynomial $M_f(\vec{x})$ that agrees with $g(\vec{x})$ on all 0-1 inputs. By Lemma 1, for any $A \subseteq \mathbb{Z}_q$ of size $Tn + 1$ we can compute an \mathcal{L}_A -restricted depth-3 circuit representation of $M_f(\vec{x})$ by evaluating $g(\vec{x})$ over the vectors $\vec{x} = 1^i 0^{Tn-i}$, which can be done using the f -oracle. \square

2.2 Lattice-Based Somewhat-Homomorphic Cryptosystems

In GGH-type [GGH97] lattice-based encryption schemes, the public key describes some lattice $L \subset \mathbb{R}^n$ and the secret key is a rational matrix $S \in \mathbb{Q}^{n \times n}$ (related to the dual lattice L^*). In the schemes that we consider, the plaintext space is \mathbb{Z}_p for a prime p , and an encryption of m is a vector $\vec{c} = \vec{v} + \vec{e} \in \mathbb{Z}^n$, where $\vec{v} \in L$ and \vec{e} is a short noise vector satisfying $\vec{e} \equiv \vec{m} \pmod{p}$. It was shown in [Gen09a] that decryption can be implemented by computing $\vec{m} \leftarrow \vec{c} - \lceil \vec{c} \cdot S \rceil \pmod{p}$, where $\lceil \cdot \rceil$ means rounding to the nearest integer. Moreover the parameters can be set to ensure that ciphertexts are close enough to the lattice so that the vector $\vec{c} \cdot S$ is less than $1/2(N + 1)$ away from \mathbb{Z}^n .

Somewhat similarly to [Gen09b], such schemes can be modified to make the secret key a bit vector $\vec{s} \in \{0, 1\}^N$, such that $S = \sum_{i=1}^N s_i \cdot T_i$ with the T_i 's public matrices. For example, the s_i 's could be the bit description of S itself, and then each T_i 's has only a single nonzero entry, of the form 2^j or 2^{-j} (for as many different values of j as needed to describe S with sufficient precision). Differently from [Gen09b], the T_i 's in our setting contain no secret information – in particular we do not require a sparse subset that sums up to S . The ciphertext \vec{c} from the original scheme is post-processed to yield $(\vec{c}, \{\vec{u}_i\}_{i=1}^N)$ where $\vec{u}_i = \vec{c} \cdot T_i$, and the decryption formula becomes $\vec{m} \leftarrow \vec{c} - \left\lceil \sum_{i=1}^N s_i \cdot \vec{u}_i \right\rceil \pmod{p}$.

Importantly, the coefficients of the \vec{u} 's are output with only $\kappa = \lceil \log(N + 1) \rceil$ bits of precision to the right of the binary point, just enough to ensure that the rounding remains correct in the decryption formula. For simplicity hereafter, we will assume that the plaintext vector is $\vec{m} = \langle 0, \dots, 0, m \rangle$ – i.e., it has only one nonzero coefficient. Thus, the post-processed ciphertext becomes $(c, \{u_i\})$ (numbers rather than vectors).

2.3 Decryption Using a Restricted Depth-3 Circuit

For the rest of this section, the details of the particular encryption scheme \mathcal{E} are irrelevant except insofar as it has the following decryption formula: The secret key is $\vec{s} \in \{0, 1\}^N$, and the ciphertext is post-processed to the form $(c, \{u_i\})$, and each u_i is split into an integer part and a fractional part, $u_i = u'_i + u''_i$, such that the fractional part has only $\kappa = \lceil \log(N + 1) \rceil$ bits of precision (namely, u''_i is a κ -bit integer). The plaintext is recovered as:

$$m \leftarrow \underbrace{c - \sum s_i \cdot u'_i}_{\text{“simple part”}} - \underbrace{\left\lceil 2^{-\kappa} \cdot \sum s_i \cdot u''_i \right\rceil}_{\text{“complicated part”}} \pmod{p}. \quad (2)$$

We now show that we can compute Equation (2) using a \mathcal{L}_A -restricted circuit.

Lemma 3. *Let p be a prime $p > 2N^2$. Regarding the “complicated part” of Equation (2), there is a univariate polynomial $f(x)$ of degree $\leq 2N^2$ such that $f(\sum s_i \cdot u''_i) = \lceil 2^{-\kappa} \cdot \sum s_i \cdot u''_i \rceil \pmod{p}$.*

Proof. Since $p > 2N^2$, there is a polynomial f of degree at most $2N^2$ such that $f(x) = \lceil 2^{-\kappa} \cdot x \rceil \pmod{p}$ for all $x \in [0, 2N^2]$. The lemma follows from the fact that $\sum s_i \cdot u''_i \in [0, N \cdot (2^\kappa - 1)] \subseteq [0, 2N^2]$. \square

Theorem 1. *Let p be a prime $p > 2N^2$. For any $A \subseteq \mathbb{Z}_p$ of cardinality at least $2N^2 + 1$, \mathcal{E} 's decryption function (Equation (2)) can be efficiently expressed as and computed using a \mathcal{L}_A -restricted depth-3 circuit C of \mathcal{L}_A -degree at most $2N^2$ having at most $2N^2 + N + 1$ product gates.*

Proof. First, consider the “complicated part”. By Lemma 3, there is a univariate polynomial $f(x)$ of degree $2N^2$ such that $f(\sum s_i \cdot u_i'') = \lceil 2^{-\kappa} \cdot \sum s_i \cdot u_i' \rceil \bmod p$. Since all $u_i'' \in \{0, \dots, 2N\}$, by Lemma 2, there is a multilinear symmetric polynomial $M_f(\vec{x})$ taking $2N^2$ inputs such that

$$f\left(\sum_i s_i \cdot u_i''\right) = M_f(s_1^{u_1''} 0^{2N-u_1''}, \dots, s_N^{u_N''} 0^{2N-u_N''})$$

for all $\vec{s} \in \{0, 1\}^N$, and moreover we can efficiently compute M_f 's representation as a \mathcal{L}_A -restricted depth-3 circuit C . By Lemma 1, C has \mathcal{L}_A -degree at most $2N^2$ and has at most $2N^2 + 1$ product gates. We have proved the theorem for the complicated part. To handle the “simple part” as an \mathcal{L}_A -restricted circuit, we can re-write it as $(c + a_1 \cdot \sum u_i') - \sum (a_1 + s_i) \cdot u_i' \bmod p$ with the constant term $\lambda_0 = (c + a_1 \cdot \sum u_i')$. The circuit for the simple part has \mathcal{L}_A -degree 1 and N “product” gates. \square

In Section 4.2, we show how to tweak the “generic” lattice-based decryption further to allow a *purely* multilinear symmetric decryption formula. (Above, only the complicated part is multilinear symmetric.) While not essential to construct leveled FHE schemes, this tweak enables interesting optimizations. For example, in 4.1 we show that we can get a very compact leveled FHE ciphertext – specifically, at one point, it consists of a *single MHE ciphertext* – e.g., a single Elgamal ciphertext! This single MHE ciphertext encrypts the value $P(a_1)$, and we show how (through a clever choice of a_i 's) to derive MHE ciphertexts that encrypt $P(a_i)$ for the other i 's.

3 Leveled FHE from SWHE and MHE

Here, we show how to take a SWHE scheme that has restricted depth-3 decryption and a MHE scheme, and combine them together into a “monstrous chimera” [Wik11] to obtain leveled FHE. The construction works much like the Elgamal-based example given in the Introduction. That is, given a SWHE ciphertext, we “decrypt” it by homomorphically evaluating its depth-3 decryption circuit, pre-processing the first level of linear polynomials $L_j(\vec{s})$ (where \vec{s} is the secret key) by encrypting them under the MHE scheme, evaluating the products under the MHE scheme, converting MHE ciphertexts into SWHE ciphertexts of the same values by evaluating the MHE's scheme's decryption function under the SWHE scheme using the encrypted MHE secret key, and finally performing the final sum (an interpolation) under the SWHE scheme. The SWHE scheme only needs to be capable of evaluating the MHE scheme's decryption circuit, followed by a quadratic polynomial. Contrary to the old blueprint, the required “homomorphic capacity” of the SWHE scheme is largely independent of the SWHE scheme's decryption function.

3.1 Notations

Recall that an encryption scheme $\mathcal{E} = (\text{KeyGen}, \text{Enc}, \text{Dec}, \text{Eval})$ with plaintext space \mathcal{P} is *somewhat-homomorphic* (SWHE) with respect to a class \mathcal{F} of multivariate functions⁴ over \mathcal{P} , if for every

⁴The class \mathcal{F} may depend on the security parameter λ .

$f(x_1, \dots, x_n) \in \mathcal{F}$ and every $m_1, \dots, m_n \in \mathcal{P}$, it holds (with probability one) that

$$\text{Dec}(sk, \text{Eval}(pk, f, c_1, \dots, c_n)) = f(m_1, \dots, m_n),$$

where (sk, pk) are generated by $\text{KeyGen}(1^\lambda)$ and the c_i 's are generated as $c_i \leftarrow \text{Enc}(pk, m_i)$. We refer to \mathcal{F} as the “homomorphic capacity” of \mathcal{E} . We say that \mathcal{E} is *multiplicatively* (resp. *additively*) homomorphic if all the functions in \mathcal{F} are naturally described as multiplication (resp. addition).

Given the encryption scheme \mathcal{E} , we denote by $\mathcal{C}_{\mathcal{E}}(pk)$ the space of “freshly-encrypted ciphertexts” for the public key pk , namely the range of the encryption function for this public key. We also denote by $\mathcal{C}_{\mathcal{E}}$ the set of freshly-encrypted ciphertexts with respect to all valid public keys, and by $\mathcal{C}_{\mathcal{E}, \mathcal{F}}$ the set of “evaluated ciphertexts” for a class of functions \mathcal{F} (i.e. those that are obtained by evaluating homomorphically a function from \mathcal{F} on ciphertexts from $\mathcal{C}_{\mathcal{E}}$). That is (for implicit security parameter λ),

$$\mathcal{C}_{\mathcal{E}} \stackrel{\text{def}}{=} \bigcup_{pk \in \text{KeyGen}} \mathcal{C}_{\mathcal{E}}(pk), \quad \mathcal{C}_{\mathcal{E}, \mathcal{F}} \stackrel{\text{def}}{=} \{\text{Eval}(pk, f, \vec{c}) : pk \in \text{KeyGen}, f \in \mathcal{F}, \vec{c} \in \mathcal{C}_{\mathcal{E}}(pk)\}$$

3.2 Compatible SWHE and MHE Schemes

To construct “chimeric” leveled FHE, the component SWHE and MHE schemes must be *compatible*:

Definition 2 (Chimerically Compatible SWHE and MHE). *Let SWHE be an encryption scheme with plaintext space \mathbb{Z}_p , which is somewhat homomorphic with respect to some class \mathcal{F} . Let MHE be a scheme with plaintext space $\mathcal{P} \subseteq \mathbb{Z}_p$, which is multiplicatively homomorphic with respect to another \mathcal{F}' .*

We say that SWHE and MHE are chimerically compatible if there exists a polynomial-size set $\mathcal{L} = \{L_j\}$ of polynomials and polynomial bounds D and B such that the following hold:

- *For every ciphertext $c \in \mathcal{C}_{\text{SWHE}, \mathcal{F}}$, the function $\mathcal{D}_c(sk) = \text{SWHE.Dec}(sk, c)$ can be evaluated by an \mathcal{L} -restricted circuit over \mathbb{Z}_p with \mathcal{L} -degree D . Moreover, an explicit description of this circuit can be computed efficiently given c .*
- *For any secret key $sk \in \text{SWHE.KeyGen}$ and any polynomial $L_j \in \mathcal{L}$ we have $L_j(sk) \in \mathcal{P}$. I.e., evaluating L_j on the secret key sk lands us in the plaintext space of MHE.*
- *The homomorphic capacity \mathcal{F}' of MHE includes all products of D or less variables.*
- *The homomorphic capacity of SWHE is sufficient to evaluate the decryption of MHE followed by a quadratic polynomial (with polynomially many terms) over \mathbb{Z}_p . Formally, the number of product gates in all the \mathcal{L} -restricted circuits from the first bullet above is at most the bound B , and for any two vectors of MHE ciphertexts $\vec{c} = \langle c_1, \dots, c_b \rangle$ and $\vec{c}' = \langle c'_1, \dots, c'_{b'} \rangle \in \mathcal{C}_{\text{MHE}, \mathcal{F}'}^{\leq B}$, the two functions*

$$\begin{aligned} \text{DAdd}_{\vec{c}, \vec{c}'}(sk) &\stackrel{\text{def}}{=} \sum_{i=1}^b \text{MHE.Dec}(sk, c_i) + \sum_{i=1}^{b'} \text{MHE.Dec}(sk, c'_i) \bmod p \\ \text{DMul}_{\vec{c}, \vec{c}'}(sk) &\stackrel{\text{def}}{=} \left(\sum_{i=1}^b \text{MHE.Dec}(sk, c_i) \right) \cdot \left(\sum_{i=1}^{b'} \text{MHE.Dec}(sk, c'_i) \right) \bmod p \end{aligned}$$

are within the homomorphic capacity of SWHE – i.e., $\text{DAdd}_{\vec{c}, \vec{c}'}(sk), \text{DMul}_{\vec{c}, \vec{c}'}(sk) \in \mathcal{F}$.

We note that the question of whether two schemes are compatible may depend crucially on the exact representation of the secret keys and ciphertexts in both. Consider for example our Elgamal instantiation from the introduction. While a naive implementation of exponentiation would have exponential degree, certainly too high to be evaluated by any known SWHE scheme, we were able to post-process the Elgamal ciphertext so as to make the degree of decryption more manageable.

We also note that we can view “additively-homomorphic encryption of discrete logarithms” as a particular type of multiplicative-homomorphic scheme, where encryption include taking discrete-logarithm (assuming that it can be done efficiently) and decryption includes exponentiation.

3.3 Chimeric Leveled FHE: The Construction

Let SWHE and MHE be chimerically compatible schemes. We construct a leveled FHE scheme as follows:

FHE.KeyGen(λ, ℓ): Takes as input the security parameter λ and the number of circuit levels ℓ that the composed scheme should be capable of evaluating. For $i \in [1, \ell]$, run

$$\left(pk_{SW}^{(i)}, sk_{SW}^{(i)} \right) \stackrel{R}{\leftarrow} \text{SWHE.KeyGen}, \quad \left(pk_{MH}^{(i)}, sk_{MH}^{(i)} \right) \stackrel{R}{\leftarrow} \text{MHE.KeyGen}.$$

Encrypt the i 'th MHE secret key under the $(i+1)$ 'st SWHE public key, $\overline{sk}_{MH}^{(i)} \leftarrow \text{SWHE.Enc}(pk_{SW}^{(i+1)}, sk_{MH}^{(i)})$. Also encrypt the i 'th SWHE secret key under the i 'th MHE public key, but in a particular format as follows: Recall that there is a polynomial-size set of polynomials \mathcal{L} such that SWHE decryption can be computed by \mathcal{L} -restricted circuits. To encrypt $sk_{SW}^{(i)}$ under $pk_{MH}^{(i)}$, compute $m_{ij} \leftarrow L_j(sk_{SW}^{(i)})$ for all $L_j \in \mathcal{L}$, and then encrypt it $c_{ij} \leftarrow \text{MHE.Enc}(pk_{MH}^{(i)}, m_{ij})$. Let $\overline{sk}_{SW}^{(i)}$ denote the collection of all the c_{ij} 's. The public key pk_{FH} consists of $(pk_{SW}^{(i)}, pk_{MH}^{(i)})$ and the encrypted secret keys $(\overline{sk}_{SW}^{(i)}, \overline{sk}_{MH}^{(i)})$ for all i . The secret key sk_{FH} consists of $sk_{SW}^{(i)}$ for all i .

FHE.Enc(pk_{FH}, m): Takes as input the public key pk_{FH} and a message in the plaintext space of the SWHE scheme. It outputs $\text{SWHE.Enc}(pk_{SW}^{(1)}, m)$.

FHE.Dec(sk_{FH}, c): Takes as input the secret key sk_{FH} and a SWHE ciphertext. Suppose the ciphertext is encrypted under $pk_{SW}^{(i)}$. It is decrypted directly using $\text{SWHE.Dec}(sk_{SW}^{(i)}, c)$.

FHE.Recrypt(pk_{FH}, c): Takes as input the public key and a ciphertext c that is a valid “evaluated SWHE ciphertext” under $pk_{SW}^{(i)}$, and outputs a “refreshed” SWHE ciphertext c' , encrypting the same plaintext but under $pk_{SW}^{(i+1)}$. It works as follows:

Circuit-generation. For a SWHE ciphertext c , generate a description of the \mathcal{L} -restricted circuit C over \mathbb{Z}_p that computes the decryption of c . Denote it by

$$C_c(sk) = \lambda_0 + \sum_{k=1}^t \lambda_k \prod_{L_j \in S_k} L_j(sk) \mod p \quad (= \text{SWHE.Dec}(sk, c))$$

Products. Pick up from the public key the encryptions under the MHE public key $pk_{MH}^{(i)}$ of the values $L_j(sk_{SW}^{(i)})$. Use the homomorphism of MHE to compute MHE encryptions of all the terms $\lambda_k \cdot \prod_{L_j \in S_k} L_j(sk_{SW}^{(i)})$. Denote the set of resulting MHE ciphertexts by c_1, \dots, c_t .

Translation. Pick up from the public key the encryption under the SWHE public key $pk_{SW}^{(i+1)}$ of the MHE secret key $sk_{MH}^{(i)}$. For each MHE ciphertext c_i from the Products step, use the homomorphism of SWHE to evaluate the function $D_{c_i}(sk) = \text{MHE.Dec}(sk, c_i)$ on the encrypted secret key. The results are SWHE ciphertexts c'_1, \dots, c'_t , where c'_j encrypts the value $\lambda_k \cdot \prod_{L_j \in S_k} L_j(sk_{SW}^{(i)})$ under $pk_{SW}^{(i+1)}$.

Summation. Use the homomorphism of SWHE to sum up all the c'_j 's and add λ_0 to get a ciphertext c^* that encrypts under $pk_{SW}^{(i+1)}$ the value

$$\lambda_0 + \sum_{k=1}^t \lambda_k \prod_{L_j \in S_k} L_j(sk_{SW}^{(i)}) \mod p = \text{SWHE.Dec}(sk_{SW}^{(i)}, c)$$

Namely, c^* encrypts under $pk_{SW}^{(i+1)}$ the same value that was encrypted in c under $pk_{SW}^{(i)}$.

FHE.Add(pk_{FH}, c_1, c_2) and FHE.Mult(pk_{FH}, c_1, c_2): Take as input the public key and two ciphertexts that are valid evaluated SWHE ciphertexts under $pk_{SW}^{(i)}$. Ciphertexts within the SWHE scheme (at any level) may be added and multiplied within the homomorphic capacity of the SWHE scheme. Once the capacity is reached, they can be decrypted and then at least one more operation can be applied.

Theorem 2. *If SWHE and MHE are chimerically compatible schemes, then the above scheme FHE is a leveled FHE scheme. Also, if both SWHE and MHE are semantically secure, then so is FHE.*

Correctness follows in a straightforward manner from the definition of chimerically compatible schemes. Security follows by a standard hybrid argument similar to Theorem 4.2.3 in [Gen09a]. We omit the details.

4 Optimizations

In the Products step of the Recrypt process (see Section 3), we compute multiple products homomorphically within the MHE scheme. In Section 4.1, we provide an optimization that allows us to compute only *a single product* in the Products step. In Section 4.2, we extend this optimization so that the entire leveled FHE ciphertext after the Products step can consist of a *single MHE ciphertext*.

4.1 Computing Only One Product

For now, let us ignore the “simple part” of our decryption function (Equation 2), which is linear and therefore does not involve any “real products”.

The products in the “complicated part” all have a special form. Specifically, by Theorem 1 and the preceding lemmas, for secret key $\vec{s} \in \{0, 1\}^N$, ciphertext $(c, \{u_i\})$, set $A \subset \mathbb{Z}_p$ with $|A| > 2N^2$, and fixed scalars $\{\lambda_j\}$ associated to a multilinear symmetric polynomial M_f , the products are all of the form $\lambda_j \cdot P(a_j)$ for all $a \in A$, where

$$P(z) = \prod_i (z + s_i)^{u''_i} \cdot (z + 0)^{2N - u''_i}.$$

We will show how to choose the a_j 's so that we can compute $P(a_j)$ for all j given only $P(a_1)$. This may seem surprising, but observe that the $P(a_j)$'s are highly redundant. Namely, if we consider the integer $v = \sum_{s_i=1} u_i''$ (which is at most $2N^2$), then we have

$$P(a_j) = (a_j + 1)^v \cdot (a_j + 0)^{2N^2-v}.$$

Knowing a_1 , the value of $P(a_1)$ contains enough information to deduce v , and then knowing a_j we can get $P(a_j)$ for all j . To be able to compute the $P(a_j)$'s efficiently from $P(a_1)$, we choose the a_j 's so that for all $j > 1$ we know integers (w_j, e_j) such that:

$$a_j = w_j \cdot a_1^{e_j} \quad \text{and} \quad a_j + 1 = w_j \cdot (a_1 + 1)^{e_j}.$$

We store (w_j, e_j) in the public key, and then compute $P(a_j) = w_j^{2N^2} \cdot P(a_1)^{e_j}$.

Importantly for our application to chimeric FHE, we can compute an encryption of $P(a_j)$ from an encryption of $P(a_1)$ within the MHE scheme – simply use the multiplicative homomorphism to exponentiate by e_j (using repeated squaring as necessary) and then multiply the result by $w_j^{2N^2}$.

Generating suitable tuples (a_j, w_j, e_j) for $j > 1$ from an initial value a_1 is straightforward: We choose the e_j 's arbitrarily and then solve for the rest. Namely, we generate distinct e_j 's, different from 0,1, then set $a_j \leftarrow a_1^{e_j} / ((a_1 + 1)^{e_j} - a_1^{e_j})$ and $w_j = a_j / a_1^{e_j}$. Observe that $a_j + 1 = (a_1 + 1)^{e_j} / ((a_1 + 1)^{e_j} - a_1^{e_j})$ – i.e., the ratio $(a_j + 1)/a_j = ((a_1 + 1)/a_1)^{e_j}$, as required.

Some care must be taken to ensure that the values $a_j, a_j + 1$ are in plaintext space of the MHE scheme – e.g., for Elgamal they need to be quadratic residues. Recall the basic fact that for a safe prime p there are $(p-3)/4$ values a for which $a, a+1 \in \text{QR}(p)$ (see Lemma 5). Therefore, finding suitable $a_1, a_1 + 1 \in \text{QR}(p)$ is straightforward. Since $a_1^{e_j}, (a_1 + 1)^{e_j} \in \text{QR}(p)$, we have

$$a_j, a_j + 1 \in \text{QR}(p) \Leftrightarrow (a_1 + 1)^{e_j} - a_1^{e_j} \in \text{QR}(p) \Leftrightarrow ((a_1 + 1)/a_1)^{e_j} - 1 \in \text{QR}(p).$$

If $(a_1 + 1)/a_1$ generates $\text{QR}(p)$ (which is certainly true if p is a safe prime), then (re-using the basic fact above) we conclude that $a_j, a_j + 1 \in \text{QR}(p)$ with probability approximately 1/2 over the choices of e_j .

Observe that the amount of extra information needed in the public key is small. The e_j 's need not be truly random – indeed, by an averaging argument over the choice of a_1 , one will quickly find an a_1 for which suitable e_j 's are $O(1)$ -dense among very small integers. Hence it is sufficient to add to the public key only $O(\log p)$ bits to specify a_1 .

4.2 Short FHE Ciphertexts: Decryption as a Pure Symmetric Polynomial

Here we provide an optimization that allows us to compress the *entire* leveled FHE ciphertext down to a *single MHE ciphertext* – e.g., a single Elgamal ciphertext! (The optimization above only compresses only representation of the “complicated part” of Equation 2, not the “simple part”.) Typically, a MHE ciphertext will be much much shorter than a SWHE ciphertext: a few thousand bits vs. millions of bits.

The main idea is that we do not need the full ciphertext $(c, \{u_i'\}, \{u_i''\})$ to recover m if we know *a priori* that m is in a small interval – e.g., $m \in \{0, 1\}$. Rather, we can choose a “large-enough” polynomial-size prime r , so that we can recover m just from $([c]_r, \{[u_i']_r\}, \{[u_i'']_r\})$, where $[x]_r$ denotes $x \bmod r \in \{0, \dots, r-1\}$. Moreover, after reducing the ciphertext components modulo r , we can invoke Lemma 2 to represent decryption as a *purely* multilinear symmetric polynomial, whose output after the product step can be represented by a *single* product $P(a_1)$ (like the complicated part in the optimization of Section 4.1).

Lemma 4. Let prime $p = \omega(N^2)$. There is a prime $r = O(N)$ and a univariate polynomial $f(x)$ of degree $O(N^2)$ such that, for all ciphertexts $(c, \{u'_i\}, \{u''_i\})$ that encrypt $m \in \{0, 1\}$, we have $m = f(t_r) \bmod p$ where

$$t_r \stackrel{\text{def}}{=} [2^\kappa \cdot c]_r + \sum_i s_i \cdot [-2^\kappa \cdot u'_i - u''_i]_r. \quad (3)$$

Proof. Let $t = 2^\kappa(c - \sum s_i \cdot u'_i) - \sum s_i \cdot u''_i$. The original decryption formula (Equation 2) is

$$m = c - \sum s_i \cdot u'_i - \lfloor 2^{-\kappa} \cdot \sum s_i \cdot u''_i \rfloor = \lfloor 2^{-\kappa} \cdot t \rfloor \bmod p$$

Thus, m can be recovered from t . Since there are only 2 possibilities for m , the (consecutive) support of t has size $2^{\kappa+1} = O(N)$. Set r to be a prime $\geq 2^{\kappa+1}$. Since the mapping $x \mapsto [x]_r$ has no collisions over the support of t , t can be recovered from $[t]_r$. Note that $[t]_r = [t_r]_r$. Thus m can be recovered from t_r (via $[t_r]_r = [t]_r$, then t). Since there are $O(N \cdot r) = O(N^2)$ possibilities for t_r , the lemma follows. \square

Theorem 3. Let prime $p = \omega(N^2)$. There is a prime $r = O(N)$ and a multilinear symmetric polynomial M such that, for all “hashed” ciphertexts $([2^\kappa \cdot c]_r, \{[-2^\kappa \cdot u'_i - u''_i]_r\})$ that encrypt $m \in \{0, 1\}$, we have

$$m = M(\underbrace{1, \dots, 1}_{[2^\kappa \cdot c]_r}, \underbrace{0, \dots, 0}_{r - [2^\kappa \cdot c]_r}, \dots, \underbrace{s_1, \dots, s_1}_{[-2^\kappa \cdot u'_1 - u''_1]_r}, \underbrace{0, \dots, 0}_{r - [-2^\kappa \cdot u'_1 - u''_1]_r}, \dots, \underbrace{s_N, \dots, s_N}_{[-2^\kappa \cdot u'_N - u''_N]_r}, \underbrace{0, \dots, 0}_{r - [-2^\kappa \cdot u'_N - u''_N]_r}) \bmod p$$

Proof. This follows easily from Lemmas 4 and 2. \square

Thus, decryption can be turned into a purely multilinear symmetric polynomial M whose product gates output $\lambda_j \cdot P(a_j)$ (for known ciphertext-independent λ_j 's), where $P(z)$ is similar to the polynomial described in Section 4.1. Using the optimization of Section 4.1, we can compress the entire leveled FHE ciphertext down to a single MHE ciphertext that encrypts $P(a_1)$.

Acknowledgments This material is based on research sponsored by DARPA under agreement number FA8750-11-C-0096. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the U.S. Government. Distribution Statement “A” (Approved for Public Release, Distribution Unlimited)

References

- [BV11] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption for ring-lwe and security for key dependent messages. In *Advances in Cryptology - CRYPTO 2011*, Lecture Notes in Computer Science. Springer, 2011.
- [ElG85] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in Cryptology - CRYPTO '84*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18. Springer-Verlag, 1985.

- [Gen09a] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. <http://crypto.stanford.edu/craig>.
- [Gen09b] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st ACM Symposium on Theory of Computing – STOC 2009*, pages 169–178. ACM, 2009.
- [Gen10] Craig Gentry. Toward basing fully homomorphic encryption on worst-case hardness. In Tal Rabin, editor, *Advances in Cryptology - CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 116–137. Springer, 2010.
- [GGH97] Oded Goldreich, Shafi Goldwasser, and Shai Halevi. Public-key cryptosystems from lattice reduction problems. In Burton S. Kaliski Jr., editor, *Advances in Cryptology - CRYPTO 1997*, volume 1294 of *Lecture Notes in Computer Science*, pages 112–131. Springer, 1997.
- [GH11] Craig Gentry and Shai Halevi. Implementing gentry’s fully-homomorphic encryption scheme. In *Advances in Cryptology - EUROCRYPT’11*, volume 6632 of *Lecture Notes in Computer Science*, pages 129–148. Springer, 2011. Full version available on-line from <http://eprint.iacr.org/2010/520>.
- [GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC’08*, pages 197–206. ACM, 2008.
- [KR] Richard M. Karp and Vijaya Ramachandran. A survey of parallel algorithms for shared-memory machines. Chapter 17 of *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity*, MIT Press, 1990, pages 869–941.
- [KS06] Adam R. Klivans and Alexander A. Sherstov. Cryptographic hardness for learning intersections of halfspaces. In *FOCS*, pages 553–562. IEEE Computer Society, 2006.
- [NW97] Noam Nisan and Avi Wigderson. Lower bounds on arithmetic circuits via partial derivatives. *Computational Complexity*, 6(3):217–234, 1997. Cites “M. Ben-Or, Private communication”.
- [Pei11] Chris Peikert, 2011. Private communication.
- [Reg04] Oded Regev. New lattice-based cryptographic constructions. *JACM*, 51(6):899–942, 2004.
- [Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *JACM*, 56(6), 2009.
- [SS10] Damien Stehlé and Ron Steinfeld. Faster fully homomorphic encryption. In Masayuki Abe, editor, *Advances in Cryptology - ASIACRYPT 2010*, volume 6477 of *Lecture Notes in Computer Science*, pages 377–394. Springer, 2010.
- [SV10] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In Phong Q. Nguyen and David Pointcheval, editors, *Public Key Cryptography - PKC 2010*, volume 6056 of *Lecture Notes in Computer Science*, pages 420–443. Springer, 2010.

[vDGHV10] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *Advances in Cryptology - EUROCRYPT'10*, volume 6110 of *Lecture Notes in Computer Science*, pages 24–43. Springer, 2010. Full version available on-line from <http://eprint.iacr.org/2009/616>.

[Wik11] Wikipedia. Chimera. <http://en.wikipedia.org/wiki/Chimera>, Accessed on August 2011.

A Instantiations of Chimeric FHE

A.1 The Homomorphic Capacity of SWHE Schemes

Our instantiations are mildly sensitive to the tradeoff between the parameters of the SWHE scheme and its homomorphic capacity. Recall that when used with plaintext space \mathbb{Z}_p , the SWHE schemes that we consider have secret key $\vec{s} \in \{0, 1\}^N$ and decryption formula⁵ for post-processed ciphertexts $(c, \{u'_i\}, \{u''_i\})$:

$$m = c + \sum_{i=1}^N s_i \cdot u'_i + \left\lceil 2^{-\kappa} \cdot \sum_{i=1}^N s_i \cdot u''_i \right\rceil \bmod p, \quad (4)$$

with $\kappa = \lceil \log(N+1) \rceil$, $u'_i \in \mathbb{Z}_p$ and $u''_i \in \{0, 1, \dots, 2^\kappa\}$. Below we say that a scheme has *threshold-type decryption* if it has this decryption formula.

We are interested in the tradeoff between the number N of secret-key bits and the degree of the polynomials that the scheme can evaluate homomorphically. For our instantiations, we only need the number of key-bits to depend polynomially on the degree. Specifically, we need a polynomial bound K , such that the scheme with plaintext space \mathbb{Z}_p with security parameter λ can be made to support α -degree polynomials with up to 2^β terms using secret keys of no more than $N = K(\lambda, \log p, \alpha, \beta)$ bits.

Below we say that a SWHE scheme is “*homomorphic for low-degree polynomials*” if it has a polynomial bound on the key-size as above. It can be verified that all the known lattice-based SWHE schemes meet this condition.

A.2 Elgamal-based Instantiation

In the Introduction, we specified (in a fair amount of detail) an instantiation of chimeric leveled FHE that uses Elgamal as the MHE scheme. Here, we provide a supporting lemmas and theorems to show that Elgamal is chimerically compatible with known SWHE schemes, as needed for the chimeric combination to actually work.

Theorem 4. *Let $p = 2q + 1$ be a safe prime such that DDH holds in $\text{QR}(p)$, and let SWHE be an encryption scheme with message space \mathbb{Z}_p , which is homomorphic for low-degree polynomials and has threshold-type decryption. Then SWHE is chimerically compatible with Elgamal encryption modulo p over plaintext space $\text{QR}(p)$.*

Proof. Denote the security parameter by λ and let $\alpha = \text{poly}(\lambda)$ be another parameter (to be set later) governing the degree of polynomials that can be homomorphically evaluated by the scheme.

⁵This formula differs from Equation (2) in that we add rather than subtract the sums. This change was done to simplify notations in some of the arguments below, and it entails only a slight modification of the scheme.

The scheme SWHE can then be set to support polynomials of degree up to α having at most 2^α terms, using a secret key $\vec{s} \in \{0, 1\}^N$ of size $N = K(\lambda, \log p, \alpha)$ for a polynomial K , with decryption formula Equation (4). Since p must be super polynomial in λ (for DDH to hold), then in particular $2N^2 < p$ and we can use Theorem 1.

We thus conclude that for any $A \subseteq \mathbb{Z}_p$ of cardinality $2N^2 + 1$, given a SWHE ciphertext $(c, \{u'_i\}, \{u''_i\})$ we can compute efficiently a \mathcal{L}_A -restricted depth-3 circuit C of \mathcal{L}_A -degree at most $2N^2$ and at most $2N^2 + N + 1$ product gates, such that $C(\vec{s}) = \text{SWHE.Dec}_{\vec{s}}(c, \{u'_i\}, \{u''_i\})$. We will thus use $D = 2N^2$ and $B = 2N^2 + N + 1$ as the bounds that are needed for Definition 2.

Next we need to establish that one can choose A so that, for any $sk \in \text{SWHE.KeyGen}$ and any polynomial $L_j \in \mathcal{L}_A$, $L_j(sk)$ is in the plaintext space of our multiplicatively homomorphic scheme. In Lemma 5 below, we show that there are $q - 1 = (p - 3)/4$ values a such that $a, a + 1 \in \text{QR}(p)$. Since N is polynomial and $2N^2 + 1 \ll q$, we can populate A with $N^2 + 1$ such values efficiently. The value $a_j + x_i$ for $a_j \in A$ and secret key bit x_i is always in $\text{QR}(p)$, which is the Elgamal plaintext space.

In this construction we trivially get the property that the MHE scheme (i.e., Elgamal) can evaluate the D multiplications needed by the circuits C , since the multiplicative homomorphic capacity of Elgamal is infinite.

It remains to show that the homomorphic capacity of the SWHE scheme is sufficient to evaluate Elgamal decryption followed by one operation (i.e., the last bullet in Definition 1). It suffices to show that Elgamal decryption can be computed using a polynomial of degree α with at most 2^α monomials, so our degree parameter α . To prepare for decryption, we post-process each Elgamal ciphertext as follows: Given a ciphertext $(y = g^r, z = m \cdot g^{-er}) \in \mathbb{Z}_p^2$, we compute $y_i = y^{2^i} - 1 \bmod p$ for $i = 0, 1, \dots, \lceil \log q \rceil - 1$, and the post-processed ciphertext is $\langle z, y_0, \dots, y_{\tau-1} \rangle$ with $\tau = \lceil \log q \rceil$. Given an Elgamal secret key $e \in \mathbb{Z}_q$ with binary representation $e_{\tau-1} \dots e_1 e_0$ (where $\tau = \lceil \log q \rceil$), decryption of the post-processed ciphertext becomes

$$\text{MHE.Dec}(e; z, y_0, \dots, y_{\tau-1}) = z \cdot \prod_{i=0}^{\tau-1} (y^{e \cdot 2^i}) = z \cdot \prod_{i=0}^{\tau-1} (e_i \cdot y_i + 1) \quad (5)$$

Being overly conservative and treating $z, y_0, \dots, y_{\tau-1}$ as variables; then the degree of the polynomial above is $2\tau + 1$, and it has 2^τ monomials. Hence the degree parameter α as $\alpha = 4 \lceil \log q \rceil + 2$, we get a scheme whose homomorphic capacity is sufficient for Elgamal decryption followed by one operation.

It remains to see that this choice of parameters is consistent. Note that the only constraints that we use in this proof are that $p = \lambda^{\omega(1)}$ (so that DDH is hard), $(p - 1)/2 = q > 2N^2 + 1 = \text{poly}(\lambda, \log q, \alpha)$ (in order to be able to use Theorem 1) and $\alpha > 4 \lceil \log q \rceil + 2$ (to get sufficient homomorphic capacity). Clearly, if p is exponential in λ (so α is polynomial in λ) then all of these constraints are satisfied. \square

Lemma 5. *Let p be a prime, and let $S = \{(X, Y) : X = Y + 1; X, Y \in \text{QR}(p)\}$. Then, $|S| = (p - 3)/4$ if $p \equiv 3 \pmod{4}$, and $|S| = (p - 5)/4$ if $p \equiv 1 \pmod{4}$.*

Proof. Let $T = \{(u, v) : u \neq 0, v \neq 0, u^2 - v^2 = 1 \bmod p\}$. Since X and Y each have exactly two nonzero square roots if they are quadratic residues, we have that $|T| = 4 \cdot |S|$. It remains to establish the cardinality of T .

For each pair $(u, v) \in T$, let $a_{uv} = u + v$. We claim that distinct pairs in T cannot have the same value of a_{uv} . In particular, each a_{uv} completely determines both u and v as follows. We

have $u^2 - v^2 = 1 \rightarrow (u - v)(u + v) = 1 \rightarrow u - v = 1/a_{uv}$, and therefore $u = (a_{uv} + a_{uv}^{-1})/2$, and $v = (a_{uv} - a_{uv}^{-1})/2$. We therefore have $|U| = |T|$, where $U = \{a \neq 0 : a + a^{-1} \neq 0, a - a^{-1} \neq 0\}$.

We have that $a \in U$, unless $a = 0$, $a^2 = -1 \pmod p$, or $a = \pm 1$. If $p = 1 \pmod 4$, then $-1 \in \text{QR}(p)$, and therefore there are 5 prohibited values of a — i.e., $|U| = p - 5$. If $p = 3 \pmod 4$, then $-1 \notin \text{QR}(p)$, and therefore $|U| = p - 3$. \square

A.3 Levelled FHE Based on Worst-Case Hardness

We next describe an instantiation where both the SWHE and the MHE schemes are lattice-based encryption schemes with security based (quantumly) on the hardness of worst-case problems over ideal lattices, in particular ideal-SIVP. This scheme could be Gentry’s SWHE scheme [Gen09b, Gen10] one of its variants [SS10, SV10, GH11], or one of the more recent proposals based on the ring-LWE problem [BV11, Pei11]. All these schemes are homomorphic for low-degree polynomials and have threshold-type decryption, in the sense of Section A.1.

The main idea of this construction is to use an *additively* homomorphic encryption (AHE) scheme (e.g., one using lattices) as our *MHE* scheme, by working with discrete logarithms. For a multiplicative group G with order q and generator g , we can view an additively homomorphic scheme AHE with plaintext space \mathbb{Z}_q as a multiplicative homomorphic scheme MHE with plaintext space G : In the MHE scheme, a ciphertext c is decrypted as $\text{MHE.Decrypt}(c) \leftarrow g^{\text{AHE.Decrypt}(c)}$. The additive homomorphism mod q thus becomes a multiplicative homomorphism in G . We can therefore use MHE as a component in chimeric levelled FHE, assuming it is compatible with a suitable SWHE scheme. One caveat is that MHE’s **Encrypt** algorithm is not obvious. Presumably, to encrypt an element $x \in G$, we encrypt its discrete log using AHE’s **Encrypt** algorithm, but this requires computing discrete logs in G . Fortunately, in our instantiation we can choose a group G of polynomial size, so computing discrete log in G can be done efficiently.

The main difficulty is to set the parameters so that the component schemes each have enough homomorphic capacity to do their jobs.

This sort of compatibility was easy for the Elgamal-based instantiation, since the parameters of the Elgamal scheme do not grow with the multiplicative homomorphic capacity required of the Elgamal scheme; Elgamal’s multiplicative homomorphic capacity is *infinite*, regardless of parameters. On the other hand, the additive homomorphic capacity of a lattice-based scheme is *limited*, as system parameters must grow (albeit slowly) to allow more additions. What makes it possible to set the parameters is the fact that such schemes can handle a *super-polynomial* number of additions.

Below let us fix some SWHE construction which is homomorphic for low-degree polynomials and has threshold-type decryption (e.g., Gentry’s scheme [Gen09b, Gen10]). For our construction we will use a polynomial-size plaintext space, namely \mathbb{Z}_p for some $p = \text{poly}(\lambda)$. In more detail, we will use two instances of the same scheme, a “large instance”, denoted **Lrg**, as the SWHE of our Chimeric construction and a “small instance”, denoted **Sml** for the MHE of our Chimeric construction. The plaintext space for **Lrg** is set as \mathbb{Z}_p for a small prime $p = \text{poly}(\lambda)$, and the plaintext space for **Sml** is set as \mathbb{Z}_q for $q = p - 1$.

We will use the small instance as a multiplicative homomorphic encryption scheme with plaintext space \mathbb{Z}_p^* . Below let g be a generator of \mathbb{Z}_p^* . Encryption of a plaintext $x \in \mathbb{Z}_p^*$ under this MHE scheme consists of first computing the discrete logarithm of x to the base g , i.e., $e \in \mathbb{Z}_q$ such that $g^e = x \pmod p$, then encrypting e under **Sml**. Similarly, MHE decryption of a ciphertext c consists of using the “native decryption” of **Sml** to recover the “native plaintext” $e \in \mathbb{Z}_q$, then exponentiating $x = g^e \pmod p$.

The homomorphic capacity of **Lrg** must be large enough to evaluate the decryption of **Sml** followed by exponentiation mod p and then a quadratic polynomial. The parameters of **Sml** can be chosen much smaller, since it only needs to support addition of polynomially many terms and not even a single multiplication.⁶

A.3.1 Decryption under Sml

The small instance has n bits of secret key, where n is some parameter to be determined later (selected to support large enough homomorphic capacity to evaluate linear polynomials with polynomially many terms.) Since native decryption of **Sml** is of the form of Equation (4), decryption under the MHE scheme has the following formula

$$\text{MHE.Dec}_{sk}(c) = g^c \cdot g^{\sum_{i=1}^n u'_i s_i} \cdot g^{\lceil 2^{-\kappa} \sum_{i=1}^n u''_i s_i \rceil} \mod p \quad (6)$$

where $(c, \{u'_i, u''_i\})$ is the post-processed ciphertext (with $u'_i \in \mathbb{Z}_q$ and $u''_i \in \mathbb{Z}_{2^\kappa}$, and $\kappa = \lceil \log(n+1) \rceil$). Below we show how this formula can be evaluated as a rather low-degree arithmetic circuit.

The complicated part. To evaluate the “complicated part”, $\lceil 2^{-\kappa} \sum_{i=1}^n u''_i s_i \rceil$, as an arithmetic circuit mod p (with input the bits s_i), we will construct a mod- p circuit that outputs the binary representation of the sum. We have n binary numbers, each with κ bits, and we need to add them over the integers and then ignore the lower κ bits. Certainly, each bit of the result can be expressed mod- p as a multilinear polynomial of degree only $n \cdot \kappa$ over the $n \cdot \kappa$ bits of the addends. It is challenging, however, to show that these low-degree representations can actually be computed efficiently.

In any case, we can compute the sum using polynomials of degree $n \cdot \kappa^c$ for small c , easily as follows: Consider a single column $\vec{x} \in \{0, 1\}^n$ of the sum. Each bit in the binary representation of the Hamming weight of \vec{x} can be expressed as a mod- p multilinear symmetric polynomial of degree n over \vec{x} . After using degree n to obtain the binary representation of the Hamming weight of each column, it only remains to add the κ κ -bit Hamming weights together (each Hamming weight shifted appropriately depending on the significance of its associated column) using degree only κ^c . Adding numbers κ κ -bit numbers is in NC^1 , and in particular can be accomplished with low degree using the “3-for-2” trick (see [KR]), repeatedly replacing each three addends by two addends that correspond to the XOR and CARRY (and hence have the same sum), each replacement only costing constant degree, and finally summing the final two addends directly. Over \mathbb{Z}_p , the 3-for-2 trick is done using the formulas

$$\begin{aligned} \text{XOR}(x, y, z) &= 4xyz - 2(xy + xz + yz) + x + y + z \\ \text{CARRY}(x, y, z) &= xy + xz + yz - 2xyz \end{aligned}$$

The simple part and exponentiation. Although it is possible to compute the simple part similarly to the complicated part, it is easier to just push this computation into the exponentiation step. Specifically, we now have a κ -bit number v_0 that we obtained as the result of the “complicated part”, and we also have the $\lceil \log q \rceil$ -bit numbers $v_i = u'_i s_i$ for $i = 1, \dots, n$ (all represented in

⁶The “small” scheme could also be instantiated from other additively homomorphic lattice-based schemes, e.g., one of Regev’s schemes [Reg04, Reg09], or the GPV scheme [GPV08], etc.

binary), and we want to compute $g^c \cdot g^{\sum_{i=0}^n v_i} \cdot \text{mod } p$. Denote the binary representation of each v_i by $(v_{it} \dots v_{i1} v_{i0})$, namely $v_i = \sum_j v_{ij} 2^j$. Then we compute

$$\begin{aligned} g^{c+(\sum_{i=0}^n v_i)} &= g^{c+(\sum_{i,j} v_{ij} 2^j)} = g^c \cdot \prod_{i,j} (g^{2^j})^{v_{ij}} = g^c \cdot \prod_{i,j} (v_{i,j} \cdot g^{2^j} + (1 - v_{i,j}) \cdot 1) \\ &= \underbrace{\prod_{j=0}^{\kappa} (1 + v_{0,j} \cdot (g^{2^j} - 1))}_{\text{"complicated part"}} \cdot \underbrace{g^c \cdot \prod_{i=1}^n \prod_{j=0}^{\lceil \log q \rceil} (1 + v_{i,j} \cdot (g^{2^j} - 1))}_{\text{"simple part"}} \end{aligned}$$

The terms g^c and $(g^{2^j} - 1)$ are known constants in \mathbb{Z}_p , hence we have a representation of the decryption formula as an arithmetic circuit mod p .

To bound the degree of the complicated part, notice that v_0 has κ bits, each a polynomial of degree at most $n \cdot \text{poly}(\kappa)$, hence the entire term has degree bounded by $n \cdot \text{poly}(\kappa)$. For the simple part, all the v_i 's together have $n \lceil \log q \rceil$ bits (each is just a variable), so the degree of that term is bounded by just $n \lceil \log q \rceil$. Hence the total degree of the decryption formula is $\tilde{O}(n)$, assuming q is quasi-polynomial in n . One can also verify that the number of terms is $2^{\tilde{O}(n)}$. (Known lattice-based SWHE schemes have $n = \tilde{O}(\lambda)$, in which case Sml's decryption has degree $\tilde{O}(\lambda)$.)

A.3.2 The SWHE scheme Lrg.

The large instance has N bits of secret key, where N is some parameter to be determined later, selected to support large enough homomorphic capacity to be compatible with Sml. As explained in Section 2, the decryption of Lrg can be expressed as a restricted depth-3 circuit of degree at most $2N^2$ and with at most $2N^2 + N + 1$ product gates. Note that the number of summands in the top addition is at most $2N^2 + N + 1 < 3N^2$.

A.3.3 Setting the parameters.

Lemma 6. *Let Lrg and Sml be as above. We can choose the parameters of Lrg and Sml so that Lrg is chimerically compatible with the MHE derived from Sml.*

Proof. Denoting the security parameter by λ , below we choose the plaintext spaces and parameters α, β , where Lrg can support polynomials of degree up to α with 2^α terms, Sml can support linear polynomials with up to 2^β terms, so as to get chimerically compatible schemes. Note that making the plaintext spaces of the two schemes compatible is simple, all we need to do is choose a prime p and set $q = p - 1$, and let the plaintext spaces of Lrg, Sml be \mathbb{Z}_p and \mathbb{Z}_q , respectively. In terms of size constraints on the parameters, we have the following:

- $p > 2N^2$, so that we can use Theorem 1.
- $p = \text{poly}(\lambda)$, so that we can compute discrete logs modulo p efficiently.
- $\beta \geq \log(2N^2) = 2 \log N + 1$, since the restricted depth-3 circuits for the decryption of Lrg all have degree at most $2N^2$, hence we need an MHE scheme that supports $2N^2$ products, which means that Sml should support linear functions with $2N^2$ terms.

- α is at least twice the degree of Sml 's decryption, so that we can compute a multiplication within Lrg after evaluating Sml 's decryption function.

Up front, we are promised polynomial bounds $K_{\text{Sml}}, K_{\text{Lrg}}$ such that the key-size of Sml is bounded by $n \leq K_{\text{Sml}}(\lambda, \log q, \beta)$ and the key-size of Lrg is bounded by $N \leq K_{\text{Lrg}}(\lambda, \log p, \alpha)$.

Assuming $N = \text{poly}(\lambda)$ (we establish this later), we can meet the first three constraints by choosing a prime $p \in [2N^2 + 1, 4N^2]$ (such a prime must exist and can be found efficiently) and $\beta = \log p$. Then $K_{\text{Sml}}(\lambda, \log q, \beta) = o(\lambda^{c_{\text{Sml}} + \epsilon})$ for any $\epsilon > 0$ and some constant c_{Sml} . We argued that before that when Sml has n -bit keys, decryption can be computed with degree $\tilde{O}(n \cdot (\log^{c_2} n + \log q))$ for some constant c_2 . Therefore, still assuming that $N = \text{poly}(\lambda)$, all of the constraints can be satisfied with $\alpha = \theta(\lambda^{c_{\text{Sml}} + \epsilon})$ for any $\epsilon > 0$. But then of course N can be $\text{poly}(\lambda)$ since it is bounded by $K_{\text{Lrg}}(\lambda, \log p, \alpha)$. \square

Using Gentry's scheme and proof [Gen09b, Gen10] we get:

Corollary 1. *There exists a leveled FHE, whose security is reducible via quantum reduction to the worst-case hardness of $S(I)VP$ in ideal lattices, ideal-SIVP. ■*

B Proof of Lemma 1

Proof. (Lemma 1) Every multilinear symmetric polynomial $M(\vec{x})$ is a linear combination of the elementary symmetric polynomials: $M(\vec{x}) = \sum_{i=0}^n \ell_i \cdot e_i(\vec{x})$. Given the evaluation $M(\vec{x})$ over binary vectors $\vec{x} = 1^i 0^{n-i}$, we can compute the ℓ_i 's as follows. We obtain the constant term $\ell_0 \cdot e_0(\vec{x}) = \ell_0$ by evaluating M at 0^n . We obtain ℓ_k recursively via

$$\begin{aligned} M(1^k 0^{n-k}) &= \sum_{i=0}^n \ell_i \cdot e_i(1^k 0^{n-k}) = \ell_k + \sum_{i=0}^{k-1} \ell_i \cdot e_i(1^k 0^{n-k}) \\ \Rightarrow \quad \ell_k &= M(1^k 0^{n-k}) - \sum_{i=0}^{k-1} \ell_i \cdot e_i(1^k 0^{n-k}) = M(1^k 0^{n-k}) - \sum_{i=0}^{k-1} \ell_i \cdot \binom{k}{i} \end{aligned}$$

At this point, it suffices to prove the lemma just for the elementary symmetric polynomials. This is because we have shown that we can efficiently obtain a representation of $M(\vec{x})$ as a linear combination of the elementary symmetric polynomials, and we can clearly use the known ℓ_j values to “merge” together the depth-3 representations of the elementary symmetric polynomials that satisfy the constraints of Lemma 1 into a depth-3 representation of M that satisfies the constraints.

For each i , the value $e_i(\vec{x})$ is the coefficient of z^{n-i} in the polynomial $P(z)$. We can compute the coefficients of $P(z)$ via interpolation from the values $P(a)$, $a \in A$. Therefore, each value $e_i(\vec{x})$ can be computed by a \mathcal{L}_A -restricted depth-3 arithmetic circuit as follows: using $n+1$ product gates, compute the values $P(a)$, $a \in A$, and then (as the final sum gate), interpolate the coefficient of z^{n-i} from the $P(a)$ values. \square

Fully Homomorphic Encryption without Bootstrapping

Zvika Brakerski
Weizmann Institute of Science

Craig Gentry*
IBM T.J. Watson Research Center

Vinod Vaikuntanathan†
University of Toronto

Abstract

We present a radically new approach to fully homomorphic encryption (FHE) that dramatically improves performance and bases security on weaker assumptions. A central conceptual contribution in our work is a new way of constructing leveled fully homomorphic encryption schemes (capable of evaluating arbitrary polynomial-size circuits), *without Gentry's bootstrapping procedure*.

Specifically, we offer a choice of FHE schemes based on the learning with error (LWE) or ring-LWE (RLWE) problems that have 2^λ security against known attacks. For RLWE, we have:

- A leveled FHE scheme that can evaluate L -level arithmetic circuits with $\tilde{O}(\lambda \cdot L^3)$ per-gate computation – i.e., computation *quasi-linear* in the security parameter. Security is based on RLWE for an approximation factor exponential in L . This construction does not use the bootstrapping procedure.
- A leveled FHE scheme that uses bootstrapping *as an optimization*, where the per-gate computation (which includes the bootstrapping procedure) is $\tilde{O}(\lambda^2)$, *independent of L* . Security is based on the hardness of RLWE for *quasi-polynomial* factors (as opposed to the sub-exponential factors needed in previous schemes).

We obtain similar results for LWE, but with worse performance. We introduce a number of further optimizations to our schemes. As an example, for circuits of large width – e.g., where a constant fraction of levels have width at least λ – we can reduce the per-gate computation of the bootstrapped version to $\tilde{O}(\lambda)$, independent of L , by *batching the bootstrapping operation*. Previous FHE schemes all required $\tilde{\Omega}(\lambda^{3.5})$ computation per gate.

At the core of our construction is a much more effective approach for managing the noise level of lattice-based ciphertexts as homomorphic operations are performed, using some new techniques recently introduced by Brakerski and Vaikuntanathan (FOCS 2011).

*Sponsored by the Air Force Research Laboratory (AFRL). Disclaimer: This material is based on research sponsored by DARPA under agreement number FA8750-11-C-0096 and FA8750-11-2-0225. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the U.S. Government. Approved for Public Release, Distribution Unlimited.

†This material is based on research sponsored by DARPA under Agreement number FA8750-11-2-0225. All disclaimers as above apply.

1 Introduction

Ancient History. Fully homomorphic encryption (FHE) [19, 8] allows a worker to receive encrypted data and perform arbitrarily-complex dynamically-chosen computations on that data while it remains encrypted, despite not having the secret decryption key. Until recently, all FHE schemes [8, 6, 20, 10, 5, 4] followed the same blueprint, namely the one laid out in Gentry’s original construction [8, 7].

The first step in Gentry’s blueprint is to construct a *somewhat homomorphic encryption (SWHE) scheme*, namely an encryption scheme capable of evaluating “low-degree” polynomials homomorphically. Starting with Gentry’s original construction based on ideal lattices [8], there are by now a number of such schemes in the literature [6, 20, 10, 5, 4, 13], all of which are based on lattices (either directly or implicitly). The ciphertexts in all these schemes are “noisy”, with a noise that grows slightly during homomorphic addition, and explosively during homomorphic multiplication, and hence, the limitation of low-degree polynomials.

To obtain FHE, Gentry provided a remarkable *bootstrapping theorem* which states that given a SWHE scheme that can evaluate its own decryption function (plus an additional operation), one can transform it into a “leveled”¹ FHE scheme. Bootstrapping “refreshes” a ciphertext by running the decryption function on it homomorphically, using an encrypted secret key (given in the public key), resulting in a reduced noise.

As if by a strange law of nature, SWHE schemes tend to be incapable of evaluating their own decryption circuits (plus some) without significant modifications. (We discuss recent exceptions [9, 3] below.) Thus, the final step is to *squash the decryption circuit* of the SWHE scheme, namely transform the scheme into one with the same homomorphic capacity but a decryption circuit that is simple enough to allow bootstrapping. Gentry [8] showed how to do this by adding a “hint” – namely, a large set with a secret sparse subset that sums to the original secret key – to the public key and relying on a “sparse subset sum” assumption.

1.1 Efficiency of Fully Homomorphic Encryption

The efficiency of fully homomorphic encryption has been a (perhaps, *the*) big question following its invention. In this paper, we are concerned with the *per-gate computation overhead* of the FHE scheme, defined as the ratio between the time it takes to compute a circuit homomorphically to the time it takes to compute it in the clear.² Unfortunately, FHE schemes that follow Gentry’s blueprint (some of which have actually been implemented [10, 5]) have fairly poor performance – their per-gate computation overhead is $p(\lambda)$, a large polynomial in the security parameter. In fact, we would like to argue that this penalty in performance is somewhat inherent for schemes that follow this blueprint.

First, the complexity of (known approaches to) bootstrapping is *inherently* at least the complexity of decryption *times* the bit-length of the individual ciphertexts that are used to encrypt the bits of the secret key. The reason is that bootstrapping involves evaluating the decryption circuit *homomorphically* – that is, in the decryption circuit, each secret-key bit is replaced by a (large) ciphertext that encrypts that bit – and both the complexity of decryption and the ciphertext lengths must each be $\Omega(\lambda)$.

Second, the undesirable properties of known SWHE schemes conspire to ensure that *the real cost of bootstrapping for FHE schemes that follow this blueprint is actually much worse than quadratic*. Known FHE schemes start with a SWHE scheme that can evaluate polynomials of degree D (multiplicative depth $\log D$) securely only if the underlying lattice problem is hard to 2^D -approximate in 2^λ time. For this to be hard, the lattice must have dimension $\Omega(D \cdot \lambda)$.³ Moreover, the coefficients of the vectors used in the

¹In a “leveled” FHE scheme, the size of the public key is linear in the *depth* of the circuits that the scheme can evaluate. One can obtain a “pure” FHE scheme (with a constant-size public key) from a leveled FHE scheme by assuming “circular security” – namely, that it is safe to encrypt the leveled FHE secret key under its own public key. We will omit the term “leveled” in this work.

²Other measures of efficiency, such as ciphertext/key size and encryption/decryption time, are also important. In fact, the schemes we present in this paper are very efficient in these aspects (as are the schemes in [9, 3]).

³This is because we have lattice algorithms in n dimensions that compute $2^{n/\lambda}$ -approximations of short vectors in time $2^{\tilde{O}(\lambda)}$.

scheme have bit length $\Omega(D)$ to allow the ciphertext noise room to expand to 2^D . Therefore, the size of “fresh” ciphertexts (e.g., those that encrypt the bits of the secret key) is $\tilde{\Omega}(D^2 \cdot \lambda)$. Since the SWHE scheme must be “bootstrappable” – i.e., capable of evaluating its own decryption function – D must exceed the degree of the decryption function. Typically, the degree of the decryption function is $\Omega(\lambda)$. Thus, overall, “fresh” ciphertexts have size $\tilde{\Omega}(\lambda^3)$. So, the real cost of bootstrapping – even if we optimistically assume that the “stale” ciphertext that needs to be refreshed can be decrypted in only $\Theta(\lambda)$ -time – is $\tilde{\Omega}(\lambda^4)$.

The analysis above ignores a nice optimization by Stehlé and Steinfeld [22], which so far has not been useful in practice, that uses Chernoff bounds to asymptotically reduce the decryption degree down to $O(\sqrt{\lambda})$. With this optimization, the per-gate computation of FHE schemes that follow the blueprint is $\tilde{\Omega}(\lambda^3)$.⁴

Recent Deviations from Gentry’s Blueprint, and the Hope for Better Efficiency. Recently, Gentry and Halevi [9], and Brakerski and Vaikuntanathan [3], independently found very different ways to construct FHE without using the squashing step, and thus without the sparse subset sum assumption. These schemes are the first major deviations from Gentry’s blueprint for FHE. Brakerski and Vaikuntanathan [3] manage to base security entirely on LWE (for sub-exponential approximation factors), avoiding reliance on ideal lattices.

From an efficiency perspective, however, these results are not a clear win over previous schemes. Both of the schemes still rely on the problematic aspects of Gentry’s blueprint – namely, bootstrapping and an SWHE scheme with the undesirable properties discussed above. Thus, their per-gate computation is still $\tilde{\Omega}(\lambda^4)$ (in fact, that is an optimistic evaluation of their performance). Nevertheless, the techniques introduced in these recent constructions are very interesting and useful to us. In particular, we use the tools and techniques introduced by Brakerski and Vaikuntanathan [3] in an essential way to achieve remarkable efficiency gains.

An important, somewhat orthogonal question is the strength of assumptions underlying FHE schemes. All the schemes so far rely on the hardness of short vector problems on lattices with a *subexponential* approximation factor. Can we base FHE on polynomial hardness assumptions?

1.2 Our Results and Techniques

We leverage Brakerski and Vaikuntanathan’s techniques [3] to achieve asymptotically very efficient FHE schemes. Also, we base security on lattice problems with *quasi-polynomial* approximation factors. (Previous schemes all used sub-exponential factors.) In particular, we have the following theorem (informal):

- Assuming Ring LWE for an approximation factor exponential in L , we have a leveled FHE scheme that can evaluate L -level arithmetic circuits *without using bootstrapping*. The scheme has $\tilde{O}(\lambda \cdot L^3)$ per-gate computation (namely, *quasi-linear* in the security parameter).
- Alternatively, assuming Ring LWE is hard for *quasi-polynomial* factors, we have a leveled FHE scheme that uses bootstrapping *as an optimization*, where the per-gate computation (which includes the bootstrapping procedure) is $\tilde{O}(\lambda^2)$, *independent of L* .

We can alternatively base security on LWE, albeit with worse performance. We now sketch our main idea for boosting efficiency.

In the BV scheme [3], like ours, a ciphertext vector $\mathbf{c} \in R^n$ (where R is a ring, and n is the “dimension” of the vector) that encrypts a message m satisfies the decryption formula $m = [[\langle \mathbf{c}, \mathbf{s} \rangle]_q]_2$, where $\mathbf{s} \in R^n$ is the secret key vector, q is an odd modulus, and $[\cdot]_q$ denotes reduction into the range $(-q/2, q/2)$. This is an abstract scheme that can be instantiated with either LWE or Ring LWE – in the LWE instantiation, R is the ring of integers mod q and n is a large dimension, whereas in the Ring LWE instantiation, R is the ring of polynomials over integers mod q and an irreducible $f(x)$, and the dimension $n = 1$.

⁴We note that bootstrapping lazily – i.e., applying the refresh procedure only at a $1/k$ fraction of the circuit levels for $k > 1$ – cannot reduce the per-gate computation further by more than a logarithmic factor for schemes that follow this blueprint, since these SWHE schemes can evaluate only \log multiplicative depth before it becomes *absolutely necessary* to refresh – i.e., $k = O(\log \lambda)$.

We will call $[\langle \mathbf{c}, \mathbf{s} \rangle]_q$ the *noise* associated to ciphertext \mathbf{c} under key \mathbf{s} . Decryption succeeds as long as the magnitude of the noise stays smaller than $q/2$. Homomorphic addition and multiplication increase the noise in the ciphertext. Addition of two ciphertexts with noise at most B results in a ciphertext with noise at most $2B$, whereas multiplication results in a noise as large as B^2 .⁵ We will describe a *noise-management technique* that keeps the noise in check by reducing it after homomorphic operations, without bootstrapping.

The key technical tool we use for noise management is the “modulus switching” technique developed by Brakerski and Vaikuntanathan [3]. Jumping ahead, we note that while they use modulus switching in “one shot” to obtain a small ciphertext (to which they then apply Gentry’s bootstrapping procedure), we will use it (iteratively, gradually) to keep the noise level essentially constant, while stingily sacrificing modulus size and gradually sacrificing the remaining homomorphic capacity of the scheme.

Modulus Switching. The essence of the modulus-switching technique is captured in the following lemma. In words, the lemma says that an evaluator, who does not know the secret key \mathbf{s} but instead only knows a bound on its length, can transform a ciphertext \mathbf{c} modulo q into a different ciphertext modulo p while preserving correctness – namely, $[\langle \mathbf{c}', \mathbf{s} \rangle]_p = [\langle \mathbf{c}, \mathbf{s} \rangle]_q \bmod 2$. The transformation from \mathbf{c} to \mathbf{c}' involves simply scaling by (p/q) and rounding appropriately! Most interestingly, if \mathbf{s} is short and p is sufficiently smaller than q , the “noise” in the ciphertext actually decreases – namely, $|\langle \mathbf{c}', \mathbf{s} \rangle|_p < |\langle \mathbf{c}, \mathbf{s} \rangle|_q$.

Lemma 1. *Let p and q be two odd moduli, and let \mathbf{c} be an integer vector. Define \mathbf{c}' to be the integer vector closest to $(p/q) \cdot \mathbf{c}$ such that $\mathbf{c}' = \mathbf{c} \bmod 2$. Then, for any \mathbf{s} with $|\langle \mathbf{c}, \mathbf{s} \rangle|_q < q/2 - (q/p) \cdot \ell_1(\mathbf{s})$, we have*

$$[\langle \mathbf{c}', \mathbf{s} \rangle]_p = [\langle \mathbf{c}, \mathbf{s} \rangle]_q \bmod 2 \quad \text{and} \quad |\langle \mathbf{c}', \mathbf{s} \rangle|_p < (p/q) \cdot |\langle \mathbf{c}, \mathbf{s} \rangle|_q + \ell_1(\mathbf{s})$$

where $\ell_1(\mathbf{s})$ is the ℓ_1 -norm of \mathbf{s} .

Proof. For some integer k , we have $[\langle \mathbf{c}, \mathbf{s} \rangle]_q = \langle \mathbf{c}, \mathbf{s} \rangle - kq$. For the same k , let $e_p = \langle \mathbf{c}', \mathbf{s} \rangle - kp \in \mathbb{Z}$. Since $\mathbf{c}' = \mathbf{c}$ and $p = q \bmod 2$, we have $e_p = [\langle \mathbf{c}, \mathbf{s} \rangle]_q \bmod 2$. Therefore, to prove the lemma, it suffices to prove that $e_p = [\langle \mathbf{c}', \mathbf{s} \rangle]_p$ and that it has small enough norm. We have $e_p = (p/q)[\langle \mathbf{c}, \mathbf{s} \rangle]_q + \langle \mathbf{c}' - (p/q)\mathbf{c}, \mathbf{s} \rangle$, and therefore $|e_p| \leq (p/q)|[\langle \mathbf{c}, \mathbf{s} \rangle]_q| + \ell_1(\mathbf{s}) < p/2$. The latter inequality implies $e_p = [\langle \mathbf{c}', \mathbf{s} \rangle]_p$. \square

Amazingly, this trick permits the evaluator to reduce the magnitude of the noise without knowing the secret key, and without bootstrapping. In other words, modulus switching gives us a very powerful and lightweight way to *manage the noise* in FHE schemes! In [3], the modulus switching technique is bundled into a “dimension reduction” procedure, and we believe it deserves a separate name and close scrutiny. It is also worth noting that our use of modulus switching does not require an “evaluation key”, in contrast to [3].

Our New Noise Management Technique. At first, it may look like modulus switching is not a very effective noise management tool. If p is smaller than q , then of course modulus switching may reduce the magnitude of the noise, but it reduces the modulus size by essentially the same amount. In short, the ratio of the noise to the “noise ceiling” (the modulus size) does not decrease at all. Isn’t this ratio what dictates the remaining homomorphic capacity of the scheme, and how can potentially worsening (certainly not improving) this ratio do anything useful?

In fact, it’s not just the ratio of the noise to the “noise ceiling” that’s important. The *absolute magnitude of the noise* is also important, especially in multiplications. Suppose that $q \approx x^k$, and that you have two mod- q SWHE ciphertexts with noise of magnitude x . If you multiply them, the noise becomes x^2 . After 4 levels of multiplication, the noise is x^{16} . If you do another multiplication at this point, you reduce the ratio of the noise ceiling (i.e. q) to the noise level by a huge factor of x^{16} – i.e., you reduce this gap very

⁵The noise after multiplication is in fact a bit larger than B^2 due to the additional noise from the BV “re-linearization” process. For the purposes of this exposition, it is best to ignore this minor detail.

fast. Thus, the actual magnitude of the noise impacts how fast this gap is reduced. After only $\log k$ levels of multiplication, the noise level reaches the ceiling.

Now, consider the following alternative approach. Choose a *ladder of gradually decreasing moduli* $\{q_i \approx q/x^i\}$ for $i < k$. After you multiply the two mod- q ciphertexts, switch the ciphertext to the smaller modulus $q_1 = q/x$. As the lemma above shows, the noise level of the new ciphertext (now with respect to the modulus q_1) goes from x^2 back down to x . (Let's suppose for now that $\ell_1(s)$ is small in comparison to x so that we can ignore it.) Now, when we multiply two ciphertexts (wrt modulus q_1) that have noise level x , the noise again becomes x^2 , but then we switch to modulus q_2 to reduce the noise back to x . In short, each level of multiplication only reduces the ratio (noise ceiling)/(noise level) by a factor of x (not something like x^{16}). With this new approach, we can perform about k (not just $\log k$) levels of multiplication before we reach the noise ceiling. We have just increased (without bootstrapping) the number of multiplicative levels that we can evaluate by an exponential factor!

This exponential improvement is enough to achieve leveled FHE without bootstrapping. For *any* polynomial k , we can evaluate circuits of depth k . The performance of the scheme degrades with k – e.g., we need to set $q = q_0$ to have bit length proportional to k – but it degrades only polynomially with k .

Our main observation – the key to obtaining FHE without bootstrapping – is so simple that it is easy to miss and bears repeating: We get noise reduction *automatically* via modulus switching, and by carefully calibrating our ladder of moduli $\{q_i\}$, one modulus for each circuit level, to be decreasing *gradually*, we can keep the noise level very small and essentially constant from one level to the next while only gradually sacrificing the size of our modulus until the ladder is used up. With this approach, we can efficiently evaluate arbitrary polynomial-size arithmetic circuits without resorting to bootstrapping.

Performance-wise, this scheme trounces previous (bootstrapping-based) FHE schemes (at least asymptotically; the concrete performance remains to be seen). Instantiated with ring-LWE, it can evaluate L -level arithmetic circuits with per-gate computation $\tilde{O}(\lambda \cdot L^3)$ – i.e., computation *quasi-linear* in the security parameter. Since the ratio of the largest modulus (namely, $q \approx x^L$) to the noise (namely, x) is exponential in L , the scheme relies on the hardness of approximating short vectors to within an exponential in L factor.

Bootstrapping for Better Efficiency and Better Assumptions. The per-gate computation of our FHE-without-bootstrapping scheme depends polynomially on the number of levels in the circuit that is being evaluated. While this approach is efficient (in the sense of “polynomial time”) for polynomial-size circuits, the per-gate computation may become undesirably high for very deep circuits. So, we re-introduce bootstrapping *as an optimization*⁶ that makes the per-gate computation independent of the circuit depth, and that (if one is willing to assume circular security) allows homomorphic operations to be performed indefinitely without needing to specify in advance a bound on the number of circuit levels. The main idea is that to compute arbitrary polynomial-depth circuits, it is enough to compute the decryption circuit of the scheme homomorphically. Since the decryption circuit has depth $\approx \log \lambda$, the largest modulus we need has only $\tilde{O}(\lambda)$ bits, and therefore we can base security on the hardness of lattice problems with quasi-polynomial factors. Since the decryption circuit has size $\tilde{O}(\lambda)$ for the RLWE-based instantiation, the per-gate computation becomes $\tilde{O}(\lambda^2)$ (independent of L). See Section 5 for details.

Other Optimizations. We also consider *batching* as an optimization. The idea behind batching is to pack multiple plaintexts into each ciphertext so that a function can be homomorphically evaluated on multiple inputs with approximately the same efficiency as homomorphically evaluating it on one input.

⁶We are aware of the seeming irony of trumpeting “FHE without bootstrapping” and then proposing bootstrapping “as an optimization”. First, FHE without bootstrapping is exciting theoretically, independent of performance. Second, whether bootstrapping actually improves performance depends crucially on the number of levels in the circuit one is evaluating. For example, for circuits of depth sub-polynomial in the security parameter, this “optimization” will not improve performance asymptotically.

An especially interesting case is *batching the decryption function* so that multiple ciphertexts – e.g., all of the ciphertexts associated to gates at some level in the circuit – can be bootstrapped simultaneously very efficiently. For circuits of large width (say, width λ), batched bootstrapping reduces the per-gate computation in the RLWE-based instantiation to $\tilde{O}(\lambda)$, independent of L . We give the details in Section 5.

1.3 Other Related Work

We note that prior to Gentry’s construction, there were already a few interesting homomorphic encryption schemes that could be called “somewhat homomorphic”, including Boneh-Goh-Nissim [2] (evaluates quadratic formulas using bilinear maps), (Aguilar Melchor)-Gaborit-Herranz [15] (evaluates constant degree polynomials using lattices) and Ishai-Paskin [12] (evaluates branching programs).

2 Preliminaries

Basic Notation. In our construction, we will use a ring R . In our concrete instantiations, we prefer to use either $R = \mathbb{Z}$ (the integers) or the polynomial ring $R = \mathbb{Z}[x]/(x^d + 1)$, where d is a power of 2.

We write elements of R in lowercase – e.g., $r \in R$. We write vectors in bold – e.g., $\mathbf{v} \in R^n$. The notation $\mathbf{v}[i]$ refers to the i -th coefficient of \mathbf{v} . We write the dot product of $\mathbf{u}, \mathbf{v} \in R^n$ as $\langle \mathbf{u}, \mathbf{v} \rangle = \sum_{i=1}^n \mathbf{u}[i] \cdot \mathbf{v}[i] \in R$. When R is a polynomial ring, $\|r\|$ for $r \in R$ refers to the Euclidean norm of r ’s coefficient vector. We say $\gamma_R = \max\{\|a \cdot b\| / \|a\| \|b\| : a, b \in R\}$ is the expansion factor of R . For $R = \mathbb{Z}[x]/(x^d + 1)$, the value of γ_R is at most \sqrt{d} by Cauchy-Schwarz.

For integer q , we use R_q to denote R/qR . Sometimes we will use abuse notation and use R_2 to denote the set of R -elements with binary coefficients – e.g., when $R = \mathbb{Z}$, R_2 may denote $\{0, 1\}$, and when R is a polynomial ring, R_2 may denote those polynomials that have 0/1 coefficients. When it is obvious that q is not a power of two, we will use $\lceil \log q \rceil$ to denote $1 + \lfloor \log q \rfloor$. For $a \in R$, we use the notation $[a]_q$ to refer to $a \bmod q$, with coefficients reduced into the range $(-q/2, q/2]$.

Leveled Fully Homomorphic Encryption. Most of this paper will focus on the construction of a *leveled* fully homomorphic scheme, in the sense that the parameters of the scheme depend (polynomially) on the depth of the circuits that the scheme is capable of evaluating.

Definition 1 (Leveled Fully Homomorphic Encryption [7]). *We say that a family of homomorphic encryption schemes $\{\mathcal{E}^{(L)} : L \in \mathbb{Z}^+\}$ is leveled fully homomorphic if, for all $L \in \mathbb{Z}^+$, they all use the same decryption circuit, $\mathcal{E}^{(L)}$ compactly evaluates all circuits of depth at most L (that use some specified complete set of gates), and the computational complexity of $\mathcal{E}^{(L)}$ ’s algorithms is polynomial (the same polynomial for all L) in the security parameter, L , and (in the case of the evaluation algorithm) the size of the circuit.*

2.1 The Learning with Errors (LWE) Problem

The learning with errors (LWE) problem was introduced by Regev [17]. It is defined as follows.

Definition 2 (LWE). *For security parameter λ , let $n = n(\lambda)$ be an integer dimension, let $q = q(\lambda) \geq 2$ be an integer, and let $\chi = \chi(\lambda)$ be a distribution over \mathbb{Z} . The $\text{LWE}_{n,q,\chi}$ problem is to distinguish the following two distributions: In the first distribution, one samples (\mathbf{a}_i, b_i) uniformly from \mathbb{Z}_q^{n+1} . In the second distribution, one first draws $\mathbf{s} \leftarrow \mathbb{Z}_q^n$ uniformly and then samples $(\mathbf{a}_i, b_i) \in \mathbb{Z}_q^{n+1}$ by sampling $\mathbf{a}_i \leftarrow \mathbb{Z}_q^n$ uniformly, $e_i \leftarrow \chi$, and setting $b_i = \langle \mathbf{a}_i, \mathbf{s} \rangle + e_i$. The $\text{LWE}_{n,q,\chi}$ assumption is that the $\text{LWE}_{n,q,\chi}$ problem is infeasible.*

Regev [17] proved that for certain moduli q and Gaussian error distributions χ , the $\text{LWE}_{n,q,\chi}$ assumption is true as long as certain worst-case lattice problems are hard to solve using a quantum algorithm. We state this result using the terminology of B -bounded distributions, which is a distribution over the integers where the magnitude of a sample is bounded with high probability. A definition follows.

Definition 3 (*B*-bounded distributions). A distribution ensemble $\{\chi_n\}_{n \in \mathbb{N}}$, supported over the integers, is called *B*-bounded if

$$\Pr_{e \leftarrow \chi_n} [|e| > B] = \text{negl}(n) .$$

We can now state Regev’s worst-case to average-case reduction for LWE.

Theorem 1 (Regev [17]). *For any integer dimension n , prime integer $q = q(n)$, and $B = B(n) \geq 2n$, there is an efficiently samplable *B*-bounded distribution χ such that if there exists an efficient (possibly quantum) algorithm that solves $\text{LWE}_{n,q,\chi}$, then there is an efficient quantum algorithm for solving $\tilde{O}(qn^{1.5}/B)$ -approximate worst-case SVP and gapSVP.*

Peikert [16] de-quantized Regev’s results to some extent – that is, he showed the $\text{LWE}_{n,q,\chi}$ assumption is true as long as certain worst-case lattice problems are hard to solve using a *classical* algorithm. (See [16] for a precise statement of these results.)

Applebaum et al. [1] showed that if LWE is hard for the above distribution of s , then it is also hard when s ’s coefficients are sampled according to the noise distribution χ .

2.2 The Ring Learning with Errors (RLWE) Problem

The ring learning with errors (RLWE) problem was introduced by Lyubashevsky, Peikert and Regev [14]. We will use an simplified special-case version of the problem that is easier to work with [18, 4].

Definition 4 (RLWE). *For security parameter λ , let $f(x) = x^d + 1$ where $d = d(\lambda)$ is a power of 2. Let $q = q(\lambda) \geq 2$ be an integer. Let $R = \mathbb{Z}[x]/(f(x))$ and let $R_q = R/qR$. Let $\chi = \chi(\lambda)$ be a distribution over R . The $\text{RLWE}_{d,q,\chi}$ problem is to distinguish the following two distributions: In the first distribution, one samples (a_i, b_i) uniformly from R_q^2 . In the second distribution, one first draws $s \leftarrow R_q$ uniformly and then samples $(a_i, b_i) \in R_q^2$ by sampling $a_i \leftarrow R_q$ uniformly, $e_i \leftarrow \chi$, and setting $b_i = a_i \cdot s + e_i$. The $\text{RLWE}_{d,q,\chi}$ assumption is that the $\text{RLWE}_{d,q,\chi}$ problem is infeasible.*

The RLWE problem is useful, because the well-established shortest vector problem (SVP) over ideal lattices can be reduced to it, specifically:

Theorem 2 (Lyubashevsky-Peikert-Regev [14]). *For any d that is a power of 2, ring $R = \mathbb{Z}[x]/(x^d + 1)$, prime integer $q = q(d) = 1 \bmod d$, and $B = \omega(\sqrt{d \log d})$, there is an efficiently samplable distribution χ that outputs elements of R of length at most B with overwhelming probability, such that if there exists an efficient algorithm that solves $\text{RLWE}_{d,q,\chi}$, then there is an efficient quantum algorithm for solving $d^{\omega(1)} \cdot (q/B)$ -approximate worst-case SVP for ideal lattices over R .*

Typically, to use RLWE with a cryptosystem, one chooses the noise distribution χ according to a Gaussian distribution, where vectors sampled according to this distribution have length only $\text{poly}(d)$ with overwhelming probability. This Gaussian distribution may need to be “ellipsoidal” for certain reductions to go through [14]. It has been shown for RLWE that one can equivalently assume that s is alternatively sampled from the noise distribution χ [14].

2.3 The General Learning with Errors (GLWE) Problem

The learning with errors (LWE) problem and the ring learning with errors problem RLWE are syntactically identical, aside from using different rings (\mathbb{Z} versus a polynomial ring) and different vector dimensions over those rings ($n = \text{poly}(\lambda)$ for LWE, but n is constant – namely, 1 – in the RLWE case). To simplify our presentation, we define a “General Learning with Errors (GLWE)” Problem, and describe a single “GLWE-based” FHE scheme, rather than presenting essentially the same scheme twice, once for each of our two concrete instantiations.

Definition 5 (GLWE). For security parameter λ , let $n = n(\lambda)$ be an integer dimension, let $f(x) = x^d + 1$ where $d = d(\lambda)$ is a power of 2, let $q = q(\lambda) \geq 2$ be a prime integer, let $R = \mathbb{Z}[x]/(f(x))$ and $R_q = R/qR$, and let $\chi = \chi(\lambda)$ be a distribution over R . The $\text{GLWE}_{n,f,q,\chi}$ problem is to distinguish the following two distributions: In the first distribution, one samples (\mathbf{a}_i, b_i) uniformly from R_q^{n+1} . In the second distribution, one first draws $\mathbf{s} \leftarrow R_q^n$ uniformly and then samples $(\mathbf{a}_i, b_i) \in R_q^{n+1}$ by sampling $\mathbf{a}_i \leftarrow R_q^n$ uniformly, $e_i \leftarrow \chi$, and setting $b_i = \langle \mathbf{a}_i, \mathbf{s} \rangle + e_i$. The $\text{GLWE}_{n,f,q,\chi}$ assumption is that the $\text{GLWE}_{n,f,q,\chi}$ problem is infeasible.

LWE is simply GLWE instantiated with $d = 1$. RLWE is GLWE instantiated with $n = 1$. Interestingly, as far as we know, instances of GLWE between these extremes have not been explored. One would suspect that GLWE is hard for any (n, d) such that $n \cdot d = \Omega(\lambda \log(q/B))$, where B is a bound (with overwhelming probability) on the length of elements output by χ . For fixed $n \cdot d$, perhaps GLWE gradually becomes harder as n increases (if it is true that general lattice problems are harder than ideal lattice problems), whereas increasing d is probably often preferable for efficiency.

If q is much larger than B , the associated GLWE problem is believed to be easier (i.e., there is less security). Previous FHE schemes required q/B to be sub-exponential in n or d to give room for the noise to grow as homomorphic operations (especially multiplication) are performed. In our FHE scheme without bootstrapping, q/B will be exponential in the number of circuit levels to be evaluated. However, since the decryption circuit can be evaluated in logarithmic depth, the bootstrapped version of our scheme will only need q/B to be quasi-polynomial, and we thus base security on lattice problems for quasi-polynomial approximation factors.

The GLWE assumption implies that the distribution $\{(\mathbf{a}_i, \langle \mathbf{a}_i, \mathbf{s} \rangle + t \cdot e_i)\}$ is computational indistinguishable from uniform for any t relatively prime to q . This fact will be convenient for encryption, where, for example, a message m may be encrypted as $(\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + 2e + m)$, and this fact can be used to argue that the second component of this message is indistinguishable from random.

3 (Leveled) FHE without Bootstrapping: Our Construction

The plan of this section is to present our leveled FHE-without-bootstrapping construction in modular steps. First, we describe a plain GLWE-based encryption scheme with no homomorphic operations. Next, we describe variants of the “relinearization” and “dimension reduction” techniques of [3]. Finally, in Section 3.4, we lay out our construction of FHE without bootstrapping.

3.1 Basic Encryption Scheme

We begin by presenting a basic GLWE-based encryption scheme with no homomorphic operations. Let λ be the security parameter, representing 2^λ security against known attacks. ($\lambda = 100$ is a reasonable value.)

Let $R = R(\lambda)$ be a ring. For example, one may use $R = \mathbb{Z}$ if one wants a scheme based on (standard) LWE, or one may use $R = \mathbb{Z}[x]/f(x)$ where (e.g.) $f(x) = x^d + 1$ and $d = d(\lambda)$ is a power of 2 if one wants a scheme based on RLWE. Let the “dimension” $n = n(\lambda)$, an odd modulus $q = q(\lambda)$, a “noise” distribution $\chi = \chi(\lambda)$ over R , and an integer $N = N(\lambda)$ be additional parameters of the system. These parameters come from the GLWE assumption, except for N , which is set to be larger than $(2n + 1) \log q$. Note that $n = 1$ in the RLWE instantiation. For simplicity, assume for now that the plaintext space is $R_2 = R/2R$, though larger plaintext spaces are certainly possible.

We go ahead and stipulate here – even though it only becomes important when we introduce homomorphic operations – that the noise distribution χ is set to be as small as possible. Specifically, to base security on LWE or GLWE, one must use (typically Gaussian) noise distributions with deviation *at least* some sub-linear function of d or n , and we will let χ be a noise distribution that barely satisfies that requirement. To

achieve 2^λ security against known lattice attacks, one must have $n \cdot d = \Omega(\lambda \cdot \log(q/B))$ where B is a bound on the length of the noise. Since n or d depends logarithmically on q , and since the distribution χ (and hence B) depends sub-linearly on n or d , the distribution χ (and hence B) depends sub-logarithmically on q . This dependence is weak, and one should think of the noise distribution as being essentially independent of q .

Here is a basic GLWE-based encryption scheme with no homomorphic operations:

Basic GLWE-Based Encryption Scheme:

- **E.Setup**($1^\lambda, 1^\mu, b$): Use the bit $b \in \{0, 1\}$ to determine whether we are setting parameters for a LWE-based scheme (where $d = 1$) or a RLWE-based scheme (where $n = 1$). Choose a μ -bit modulus q and choose the other parameters ($d = d(\lambda, \mu, b)$, $n = n(\lambda, \mu, b)$, $N = \lceil (2n + 1) \log q \rceil$, $\chi = \chi(\lambda, \mu, b)$) appropriately to ensure that the scheme is based on a GLWE instance that achieves 2^λ security against known attacks. Let $R = \mathbb{Z}[x]/(x^d + 1)$ and let $params = (q, d, n, N, \chi)$.
- **E.SecretKeyGen**($params$): Draw $s' \leftarrow \chi^n$. Set $sk = s \leftarrow (1, s'[1], \dots, s'[n]) \in R_q^{n+1}$.
- **E.PublicKeyGen**($params, sk$): Takes as input a secret key $sk = s = (1, s')$ with $s[0] = 1$ and $s' \in R_q^n$ and the $params$. Generate matrix $A' \leftarrow R_q^{N \times n}$ uniformly and a vector $e \leftarrow \chi^N$ and set $b \leftarrow A's' + 2e$. Set A to be the $(n + 1)$ -column matrix consisting of b followed by the n columns of $-A'$. (Observe: $A \cdot s = 2e$.) Set the public key $pk = A$.
- **E.Enc**($params, pk, m$): To encrypt a message $m \in R_2$, set $\mathbf{m} \leftarrow (m, 0, \dots, 0) \in R_q^{n+1}$, sample $\mathbf{r} \leftarrow R_2^N$ and output the ciphertext $\mathbf{c} \leftarrow \mathbf{m} + A^T \mathbf{r} \in R_q^{n+1}$.
- **E.Dec**($params, sk, \mathbf{c}$): Output $m \leftarrow \llbracket \langle \mathbf{c}, \mathbf{s} \rangle \rrbracket_2$.

Correctness is easy to see, and it is straightforward to base security on special cases (depending on the parameters) of the GLWE assumption (and one can find such proofs of special cases in prior work).

3.2 Key Switching (Dimension Reduction)

We start by reminding the reader that in the basic GLWE-based encryption scheme above, the decryption equation for a ciphertext \mathbf{c} that encrypts m under key \mathbf{s} can be written as $m = \llbracket L_{\mathbf{c}}(\mathbf{s}) \rrbracket_2$ where $L_{\mathbf{c}}(\mathbf{x})$ is a ciphertext-dependent linear equation over the coefficients of \mathbf{x} given by $L_{\mathbf{c}}(\mathbf{x}) = \langle \mathbf{c}, \mathbf{x} \rangle$.

Suppose now that we have two ciphertexts \mathbf{c}_1 and \mathbf{c}_2 , encrypting m_1 and m_2 respectively under the same secret key \mathbf{s} . The way homomorphic multiplication is accomplished in [3] is to consider the *quadratic* equation $Q_{\mathbf{c}_1, \mathbf{c}_2}(\mathbf{x}) \leftarrow L_{\mathbf{c}_1}(\mathbf{x}) \cdot L_{\mathbf{c}_2}(\mathbf{x})$. Assuming the noises of the initial ciphertexts are small enough, we obtain $m_1 \cdot m_2 = \llbracket Q_{\mathbf{c}_1, \mathbf{c}_2}(\mathbf{s}) \rrbracket_2$, as desired. If one wishes, one can view $Q_{\mathbf{c}_1, \mathbf{c}_2}(\mathbf{x})$ as a *linear* equation $L_{\mathbf{c}_1, \mathbf{c}_2}^{long}(\mathbf{x} \otimes \mathbf{x})$ over the coefficients of $\mathbf{x} \otimes \mathbf{x}$ – that is, the tensoring of \mathbf{x} with itself – where $\mathbf{x} \otimes \mathbf{x}$'s dimension is roughly the square of \mathbf{x} 's. Using this interpretation, the ciphertext represented by the coefficients of the linear equation L^{long} is decryptable by the long secret key $\mathbf{s}_1 \otimes \mathbf{s}_1$ via the usual dot product. Of course, we cannot continue increasing the dimension like this indefinitely and preserve efficiency.

Thus, Brakerski and Vaikuntanathan convert the long ciphertext represented by the linear equation L^{long} and decryptable by the long tensored secret key $\mathbf{s}_1 \otimes \mathbf{s}_1$ into a shorter ciphertext \mathbf{c}_2 that is decryptable by a different secret key \mathbf{s}_2 . (The secret keys need to be different to avoid a “circular security” issue). Encryptions of $\mathbf{s}_1 \otimes \mathbf{s}_1$ under \mathbf{s}_2 are provided in the public key as a “hint” to facilitate this conversion.

We observe that Brakerski and Vaikuntanathan’s relinearization / dimension reduction procedures are actually quite a bit more general. They can be used to not only reduce the dimension of the ciphertext, but more generally, can be used to transform a ciphertext \mathbf{c}_1 that is decryptable under one secret key vector \mathbf{s}_1 to

a different ciphertext c_2 that encrypts the same message, but is now decryptable under a second secret key vector s_2 . The vectors c_2, s_2 may not necessarily be of lower degree or dimension than c_1, s_1 .

Below, we review the concrete details of Brakerski and Vaikuntanathan's key switching procedures. The procedures will use some subroutines that, given two vectors c and s , "expand" these vectors to get longer (higher-dimensional) vectors c' and s' such that $\langle c', s' \rangle = \langle c, s \rangle \bmod q$. We describe these subroutines first.

- $\text{BitDecomp}(x \in R_q^n, q)$ decomposes x into its bit representation. Namely, write $x = \sum_{j=0}^{\lceil \log q \rceil} 2^j \cdot u_j$, where all of the vectors u_j are in R_2^n , and output $(u_0, u_1, \dots, u_{\lceil \log q \rceil}) \in R_2^{n \cdot \lceil \log q \rceil}$.
- $\text{Powersof2}(x \in R_q^n, q)$ outputs the vector $(x, 2 \cdot x, \dots, 2^{\lceil \log q \rceil} \cdot x) \in R_q^{n \cdot \lceil \log q \rceil}$.

If one knows *a priori* that x has coefficients in $[0, B]$ for $B \ll q$, then BitDecomp can be optimized in the obvious way to output a shorter decomposition in $R_2^{n \cdot \lceil \log B \rceil}$. Observe that:

Lemma 2. *For vectors c, s of equal length, we have $\langle \text{BitDecomp}(c, q), \text{Powersof2}(s, q) \rangle = \langle c, s \rangle \bmod q$.*

Proof.

$$\langle \text{BitDecomp}(c, q), \text{Powersof2}(s, q) \rangle = \sum_{j=0}^{\lceil \log q \rceil} \langle u_j, 2^j \cdot s \rangle = \sum_{j=0}^{\lceil \log q \rceil} \langle 2^j \cdot u_j, s \rangle = \left\langle \sum_{j=0}^{\lceil \log q \rceil} 2^j \cdot u_j, s \right\rangle = \langle c, s \rangle.$$

□

We remark that this obviously generalizes to decompositions wrt bases other than the powers of 2.

Now, key switching consists of two procedures: first, a procedure $\text{SwitchKeyGen}(s_1, s_2, n_1, n_2, q)$, which takes as input the two secret key vectors as input, the respective dimensions of these vectors, and the modulus q , and outputs some auxiliary information $\tau_{s_1 \rightarrow s_2}$ that enables the switching; and second, a procedure $\text{SwitchKey}(\tau_{s_1 \rightarrow s_2}, c_1, n_1, n_2, q)$, that takes this auxiliary information and a ciphertext encrypted under s_1 and outputs a new ciphertext c_2 that encrypts the same message under the secret key s_2 . (Below, we often suppress the additional arguments n_1, n_2, q .)

$\text{SwitchKeyGen}(s_1 \in R_q^{n_1}, s_2 \in R_q^{n_2})$:

1. Run $A \leftarrow E.\text{PublicKeyGen}(s_2, N)$ for $N = n_1 \cdot \lceil \log q \rceil$.
2. Set $B \leftarrow A + \text{Powersof2}(s_1)$ (Add $\text{Powersof2}(s_1) \in R_q^N$ to A 's first column.) Output $\tau_{s_1 \rightarrow s_2} = B$.

$\text{SwitchKey}(\tau_{s_1 \rightarrow s_2}, c_1)$: Output $c_2 = \text{BitDecomp}(c_1)^T \cdot B \in R_q^{n_2}$.

Note that, in SwitchKeyGen , the matrix A basically consists of encryptions of 0 under the key s_2 . Then, pieces of the key s_1 are added to these encryptions of 0. Thus, in some sense, the matrix B consists of encryptions of pieces of s_1 (in a certain format) under the key s_2 . We now establish that the key switching procedures are meaningful, in the sense that they preserve the correctness of decryption under the new key.

Lemma 3. [Correctness] *Let $s_1, s_2, q, n_1, n_2, A, B = \tau_{s_1 \rightarrow s_2}$ be as in $\text{SwitchKeyGen}(s_1, s_2)$, and let $A \cdot s_2 = 2e_2 \in R_q^{n_2}$. Let $c_1 \in R_q^{n_1}$ and $c_2 \leftarrow \text{SwitchKey}(\tau_{s_1 \rightarrow s_2}, c_1)$. Then,*

$$\langle c_2, s_2 \rangle = 2 \langle \text{BitDecomp}(c_1), e_2 \rangle + \langle c_1, s_1 \rangle \bmod q$$

Proof.

$$\begin{aligned}
\langle \mathbf{c}_2, \mathbf{s}_2 \rangle &= \text{BitDecomp}(\mathbf{c}_1)^T \cdot \mathbf{B} \cdot \mathbf{s}_2 \\
&= \text{BitDecomp}(\mathbf{c}_1)^T \cdot (2\mathbf{e}_2 + \text{Powersof2}(\mathbf{s}_1)) \\
&= 2 \langle \text{BitDecomp}(\mathbf{c}_1), \mathbf{e}_2 \rangle + \langle \text{BitDecomp}(\mathbf{c}_1), \text{Powersof2}(\mathbf{s}_1) \rangle \\
&= 2 \langle \text{BitDecomp}(\mathbf{c}_1), \mathbf{e}_2 \rangle + \langle \mathbf{c}_1, \mathbf{s}_1 \rangle
\end{aligned}$$

□

Note that the dot product of $\text{BitDecomp}(\mathbf{c}_1)$ and \mathbf{e}_2 is small, since $\text{BitDecomp}(\mathbf{c}_1)$ is in R_2^N . Overall, we have that \mathbf{c}_2 is a valid encryption of m under key \mathbf{s}_2 , with noise that is larger by a small additive factor.

3.3 Modulus Switching

Suppose \mathbf{c} is a valid encryption of m under \mathbf{s} modulo q (i.e., $m = \llbracket \langle \mathbf{c}, \mathbf{s} \rangle \rrbracket_q$), and that \mathbf{s} is a *short* vector. Suppose also that \mathbf{c}' is basically a simple scaling of \mathbf{c} – in particular, \mathbf{c}' is the R -vector closest to $(p/q) \cdot \mathbf{c}$ such that $\mathbf{c}' = \mathbf{c} \bmod 2$. Then, it turns out (subject to some qualifications) that \mathbf{c}' is a valid encryption of m under \mathbf{s} modulo p using the usual decryption equation – that is, $m = \llbracket \langle \mathbf{c}', \mathbf{s} \rangle \rrbracket_p$! In other words, we can change the inner modulus in the decryption equation – e.g., to a smaller number – while preserving the correctness of decryption under the same secret key! The essence of this modulus switching idea, a variant of Brakerski and Vaikuntanathan’s modulus reduction technique, is formally captured in Lemma 4 below.

Definition 6 (Scale). *For integer vector \mathbf{x} and integers $q > p > m$, we define $\mathbf{x}' \leftarrow \text{Scale}(\mathbf{x}, q, p, r)$ to be the R -vector closest to $(p/q) \cdot \mathbf{x}$ that satisfies $\mathbf{x}' = \mathbf{x} \bmod r$.*

Definition 7 ($\ell_1^{(R)}$ norm). *The (usual) norm $\ell_1(\mathbf{s})$ over the reals equals $\sum_i \|\mathbf{s}[i]\|$. We extend this to our ring R as follows: $\ell_1^{(R)}(\mathbf{s})$ for $\mathbf{s} \in R^n$ is defined as $\sum_i \|\mathbf{s}[i]\|$.*

Lemma 4. *Let d be the degree of the ring (e.g., $d = 1$ when $R = \mathbb{Z}$). Let $q > p > r$ be positive integers satisfying $q = p = 1 \bmod r$. Let $\mathbf{c} \in R^n$ and $\mathbf{c}' \leftarrow \text{Scale}(\mathbf{c}, q, p, r)$. Then, for any $\mathbf{s} \in R^n$ with $\|\llbracket \langle \mathbf{c}, \mathbf{s} \rangle \rrbracket_q\| < q/2 - (q/p) \cdot (r/2) \cdot \sqrt{d} \cdot \gamma(R) \cdot \ell_1^{(R)}(\mathbf{s})$, we have*

$$\llbracket \langle \mathbf{c}', \mathbf{s} \rangle \rrbracket_p = \llbracket \langle \mathbf{c}, \mathbf{s} \rangle \rrbracket_q \bmod r \quad \text{and} \quad \|\llbracket \langle \mathbf{c}', \mathbf{s} \rangle \rrbracket_p\| < (p/q) \cdot \|\llbracket \langle \mathbf{c}, \mathbf{s} \rangle \rrbracket_q\| + (r/2) \cdot \sqrt{d} \cdot \gamma(R) \cdot \ell_1^{(R)}(\mathbf{s})$$

Proof. (Lemma 4) We have

$$\llbracket \langle \mathbf{c}, \mathbf{s} \rangle \rrbracket_q = \langle \mathbf{c}, \mathbf{s} \rangle - kq$$

for some $k \in R$. For the same k , let

$$e_p = \langle \mathbf{c}', \mathbf{s} \rangle - kp \in R$$

Note that $e_p = \llbracket \langle \mathbf{c}', \mathbf{s} \rangle \rrbracket_p \bmod p$. We claim that $\|e_p\|$ is so small that $e_p = \llbracket \langle \mathbf{c}', \mathbf{s} \rangle \rrbracket_p$. We have:

$$\begin{aligned}
\|e_p\| &= \| -kp + \langle (p/q) \cdot \mathbf{c}, \mathbf{s} \rangle + \langle \mathbf{c}' - (p/q) \cdot \mathbf{c}, \mathbf{s} \rangle \| \\
&\leq \| -kp + \langle (p/q) \cdot \mathbf{c}, \mathbf{s} \rangle \| + \| \langle \mathbf{c}' - (p/q) \cdot \mathbf{c}, \mathbf{s} \rangle \| \\
&\leq (p/q) \cdot \|\llbracket \langle \mathbf{c}, \mathbf{s} \rangle \rrbracket_q\| + \gamma(R) \cdot \sum_{j=1}^n \|\mathbf{c}'[j] - (p/q) \cdot \mathbf{c}[j]\| \cdot \|\mathbf{s}[j]\| \\
&\leq (p/q) \cdot \|\llbracket \langle \mathbf{c}, \mathbf{s} \rangle \rrbracket_q\| + \gamma(R) \cdot (r/2) \cdot \sqrt{d} \cdot \ell_1^{(R)}(\mathbf{s}) \\
&< p/2
\end{aligned}$$

Furthermore, modulo r , we have $\llbracket \langle \mathbf{c}', \mathbf{s} \rangle \rrbracket_p = e_p = \langle \mathbf{c}', \mathbf{s} \rangle - kp = \langle \mathbf{c}, \mathbf{s} \rangle - kq = \llbracket \langle \mathbf{c}, \mathbf{s} \rangle \rrbracket_q$. □

The lemma implies that an evaluator, who does not know the secret key but instead only knows a bound on its length, can potentially transform a ciphertext \mathbf{c} that encrypts m under key \mathbf{s} for modulus q – i.e., $m = [[\langle \mathbf{c}, \mathbf{s} \rangle]_q]_r$ – into a ciphertext \mathbf{c} that encrypts m under the same key \mathbf{s} for modulus p – i.e., $m = [[\langle \mathbf{c}, \mathbf{s} \rangle]_p]_r$. Specifically, the following corollary follows immediately from Lemma 4.

Corollary 1. *Let p and q be two odd moduli. Suppose \mathbf{c} is an encryption of bit m under key \mathbf{s} for modulus q – i.e., $m = [[\langle \mathbf{c}, \mathbf{s} \rangle]_q]_r$. Moreover, suppose that \mathbf{s} is a fairly short key and the “noise” $e_q \leftarrow [\langle \mathbf{c}, \mathbf{s} \rangle]_q$ has small magnitude – precisely, assume that $\|e_q\| < q/2 - (q/p) \cdot (r/2) \cdot \sqrt{d} \cdot \gamma(R) \cdot \ell_1^{(R)}(\mathbf{s})$. Then $\mathbf{c}' \leftarrow \text{Scale}(\mathbf{c}, q, p, r)$ is an encryption of bit m under key \mathbf{s} for modulus p – i.e., $m = [[\langle \mathbf{c}', \mathbf{s} \rangle]_p]_r$. The noise $e_p = [\langle \mathbf{c}', \mathbf{s} \rangle]_p$ of the new ciphertext has magnitude at most $(p/q) \cdot \|\langle \mathbf{c}, \mathbf{s} \rangle\| + \gamma(R) \cdot (r/2) \cdot \sqrt{d} \cdot \ell_1^{(R)}(\mathbf{s})$.*

Amazingly, assuming p is smaller than q and \mathbf{s} has coefficients that are small in relation to q , this trick permits the evaluator to reduce the magnitude of the noise without knowing the secret key! (Of course, this is also what Gentry’s bootstrapping transformation accomplishes, but in a much more complicated way.)

3.4 (Leveled) FHE Based on GLWE without Bootstrapping

We now present our FHE scheme. Given the machinery that we have described in the previous subsections, the scheme itself is remarkably simple.

In our scheme, we will use a parameter L indicating the number of levels of arithmetic circuit that we want our FHE scheme to be capable of evaluating. Note that this is an exponential improvement over prior schemes, that would typically use a parameter d indicating the *degree* of the polynomials to be evaluated.

(Note: the linear polynomial L^{long} , used below, is defined in Section 3.2.)

Our FHE Scheme without Bootstrapping:

- $\text{FHE.Setup}(1^\lambda, 1^L, b)$: Takes as input the security parameter, a number of levels L , and a bit b . Use the bit $b \in \{0, 1\}$ to determine whether we are setting parameters for a LWE-based scheme (where $d = 1$) or a RLWE-based scheme (where $n = 1$). Let $\mu = \mu(\lambda, L, b) = \theta(\log \lambda + \log L)$ be a parameter that we will specify in detail later. For $j = L$ (input level of circuit) to 0 (output level), run $\text{params}_j \leftarrow \text{E.Setup}(1^\lambda, 1^{(j+1) \cdot \mu}, b)$ to obtain a ladder of decreasing moduli from $q_L ((L+1) \cdot \mu$ bits) down to q_0 (μ bits). For $j = L-1$ to 0, replace the value of d_j in params_j with $d = d_L$ and the distribution χ_j with $\chi = \chi_L$. (That is, the ring dimension and noise distribution do not depend on the circuit level, but the vector dimension n_j still might.)
- $\text{FHE.KeyGen}(\{\text{params}_j\})$: For $j = L$ down to 0, do the following:
 1. Run $\mathbf{s}_j \leftarrow \text{E.SecretKeyGen}(\text{params}_j)$ and $\mathbf{A}_j \leftarrow \text{E.PublicKeyGen}(\text{params}_j, \mathbf{s}_j)$.
 2. Set $\mathbf{s}'_j \leftarrow \mathbf{s}_j \otimes \mathbf{s}_j \in R_{q_j}^{\binom{n_j+1}{2}}$. That is, \mathbf{s}'_j is a tensoring of \mathbf{s}_j with itself whose coefficients are each the product of two coefficients of \mathbf{s}_j in R_{q_j} .
 3. Set $\mathbf{s}''_j \leftarrow \text{BitDecomp}(\mathbf{s}'_j, q_j)$.
 4. Run $\tau_{\mathbf{s}''_{j+1} \rightarrow \mathbf{s}_j} \leftarrow \text{SwitchKeyGen}(\mathbf{s}''_j, \mathbf{s}_{j-1})$. (Omit this step when $j = L$.)

The secret key sk consists of the \mathbf{s}_j ’s and the public key pk consists of the \mathbf{A}_j ’s and $\tau_{\mathbf{s}''_{j+1} \rightarrow \mathbf{s}_j}$ ’s.

- $\text{FHE.Enc}(\text{params}, pk, m)$: Take a message in R_2 . Run $\text{E.Enc}(\mathbf{A}_L, m)$.
- $\text{FHE.Dec}(\text{params}, sk, \mathbf{c})$: Suppose the ciphertext is under key \mathbf{s}_j . Run $\text{E.Dec}(\mathbf{s}_j, \mathbf{c})$. (The ciphertext could be augmented with an index indicating which level it belongs to.)

- $\text{FHE.Add}(pk, c_1, c_2)$: Takes two ciphertexts encrypted under the same s_j . (If they are not initially, use FHE.Refresh (below) to make it so.) Set $c_3 \leftarrow c_1 + c_2 \bmod q_j$. Interpret c_3 as a ciphertext under s'_j (s'_j 's coefficients include all of s_j 's since $s'_j = s_j \otimes s_j$ and s_j 's first coefficient is 1) and output:

$$c_4 \leftarrow \text{FHE.Refresh}(c_3, \tau_{s''_j \rightarrow s_{j-1}}, q_j, q_{j-1})$$

- $\text{FHE.Mult}(pk, c_1, c_2)$: Takes two ciphertexts encrypted under the same s_j . If they are not initially, use FHE.Refresh (below) to make it so.) First, multiply: the new ciphertext, under the secret key $s'_j = s_j \otimes s_j$, is the coefficient vector c_3 of the linear equation $L_{c_1, c_2}^{long}(x \otimes x)$. Then, output:

$$c_4 \leftarrow \text{FHE.Refresh}(c_3, \tau_{s''_j \rightarrow s_{j-1}}, q_j, q_{j-1})$$

- $\text{FHE.Refresh}(c, \tau_{s''_j \rightarrow s_{j-1}}, q_j, q_{j-1})$: Takes a ciphertext encrypted under s'_j , the auxiliary information $\tau_{s''_j \rightarrow s_{j-1}}$ to facilitate key switching, and the current and next moduli q_j and q_{j-1} . Do the following:

1. Expand: Set $c_1 \leftarrow \text{Powersof2}(c, q_j)$. (Observe: $\langle c_1, s''_j \rangle = \langle c, s'_j \rangle \bmod q_j$ by Lemma 2.)
2. Switch Moduli: Set $c_2 \leftarrow \text{Scale}(c_1, q_j, q_{j-1}, 2)$, a ciphertext under the key s''_j for modulus q_{j-1} .
3. Switch Keys: Output $c_3 \leftarrow \text{SwitchKey}(\tau_{s''_j \rightarrow s_{j-1}}, c_2, q_{j-1})$, a ciphertext under the key s_{j-1} for modulus q_{j-1} .

Remark 1. We mention the obvious fact that, since addition increases the noise much more slowly than multiplication, one does not necessarily need to refresh after additions, even high fan-in ones.

The key step of our new FHE scheme is the Refresh procedure. If the modulus q_{j-1} is chosen to be smaller than q_j by a sufficient multiplicative factor, then Corollary 1 implies that the noise of the ciphertext output by Refresh is smaller than that of the input ciphertext – that is, the ciphertext will indeed be a “refreshed” encryption of the same value. We elaborate on this analysis in the next section.

One can reasonably argue that this scheme is not “FHE without bootstrapping” since $\tau_{s''_j \rightarrow s_{j-1}}$ can be viewed as an encrypted secret key, and the SwitchKey step can be viewed as a homomorphic evaluation of the decryption function. We prefer not to view the SwitchKey step this way. While there is some high-level resemblance, the low-level details are very different, a difference that becomes tangible in the much better asymptotic performance. To the extent that it performs decryption, SwitchKey does so very efficiently using an efficient (not bit-wise) representation of the secret key that allows this step to be computed in quasi-linear time for the RLWE instantiation, below the quadratic lower bound for bootstrapping. Certainly SwitchKey does not use the usual ponderous approach of representing the decryption function as a boolean circuit to be traversed homomorphically. Another difference is that the SwitchKey step does not actually reduce the noise level (as bootstrapping does); rather, the noise is reduced by the Scale step.

4 Correctness, Setting the Parameters, Performance, and Security

Here, we will show how to set the parameters of the scheme so that the scheme is correct. Mostly, this involves analyzing each of the steps within FHE.Add and FHE.Mult – namely, the addition or multiplication itself, and then the Powersof2, Scale and SwitchKey steps that make up FHE.Refresh – to establish that the output of each step is a decryptable ciphertext with bounded noise. This analysis will lead to concrete suggestions for how to set the ladder of moduli and to asymptotic bounds on the performance of the scheme.

Let us begin by considering how much noise FHE.Enc introduces initially.

4.1 The Initial Noise from FHE.Enc

Recall that FHE.Enc simply invokes E.Enc for suitable parameters ($params_L$) that depend on λ and L . In turn, the noise of ciphertexts output by E.Enc depends on the noise of the initial “ciphertexts” (the encryptions of 0) implicit in the matrix A output by E.PublicKeyGen, whose noise distribution is dictated by the distribution χ .

Lemma 5. *Let n_L and q_L be the parameters associated to FHE.Enc. Let d be the dimension of the ring R , and let γ_R be the expansion factor associated to R . (Both of these quantities are 1 when $R = \mathbb{Z}$.) Let B_χ be a bound such that R -elements sampled from the noise distribution χ have length at most B_χ with overwhelming probability. The length of the noise in ciphertexts output by FHE.Enc is at most $1 + 2 \cdot \gamma_R \cdot \sqrt{d} \cdot ((2n_L + 1) \log q_L) \cdot B_\chi$.*

Proof. Recall that $\mathbf{s} \leftarrow \text{E.SecretKeyGen}$ and $\mathbf{A} \leftarrow \text{E.PublicKeyGen}(\mathbf{s}, N)$ for $N = (2n_L + 1) \log q_L$, where $\mathbf{A} \cdot \mathbf{s} = 2\mathbf{e}$ for $\mathbf{e} \leftarrow \chi$. Recall that encryption works as follows: $\mathbf{c} \leftarrow \mathbf{m} + \mathbf{A}^T \mathbf{r} \bmod q$ where $\mathbf{r} \in R_2^N$. We have that the noise of this ciphertext is $[\langle \mathbf{c}, \mathbf{s} \rangle]_q = [m + 2\langle \mathbf{r}, \mathbf{e} \rangle]_q$, whose magnitude is at most $1 + 2 \cdot \gamma_R \cdot \sum_{j=1}^N \|\mathbf{r}[j]\| \cdot \|\mathbf{e}[j]\| \leq 1 + 2 \cdot \gamma_R \cdot \sqrt{d} \cdot N \cdot B_\chi$. \square

Notice that we are using very loose (i.e., conservative) upper bounds for the noise. These bounds could be tightened up with a more careful analysis. The correctness of decryption for ciphertexts output by FHE.Enc, assuming the noise bound above is less than $q/2$, follows directly from the correctness of the basic encryption and decryption algorithms E.Enc and E.Dec.

4.2 Correctness and Performance of FHE.Add and FHE.Mult (before FHE.Refresh)

Consider FHE.Mult. One begins FHE.Mult($pk, \mathbf{c}_1, \mathbf{c}_2$) with two ciphertexts under key \mathbf{s}_j for modulus q_j that have noises $e_i = [L_{\mathbf{c}_i}(\mathbf{s}_j)]_{q_j}$, where $L_{\mathbf{c}_i}(\mathbf{x})$ is simply the dot product $\langle \mathbf{c}_i, \mathbf{x} \rangle$. To multiply together two ciphertexts, one multiplies together these two linear equations to obtain a quadratic equation $Q_{\mathbf{c}_1, \mathbf{c}_2}(\mathbf{x}) \leftarrow L_{\mathbf{c}_1}(\mathbf{x}) \cdot L_{\mathbf{c}_2}(\mathbf{x})$, and then interprets this quadratic equation as a linear equation $L_{\mathbf{c}_1, \mathbf{c}_2}^{\text{long}}(\mathbf{x} \otimes \mathbf{x}) = Q_{\mathbf{c}_1, \mathbf{c}_2}(\mathbf{x})$ over the tensored vector $\mathbf{x} \otimes \mathbf{x}$. The coefficients of this long linear equation compose the new ciphertext vector \mathbf{c}_3 . Clearly, $[\langle \mathbf{c}_3, \mathbf{s}_j \otimes \mathbf{s}_j \rangle]_{q_j} = [L_{\mathbf{c}_1, \mathbf{c}_2}^{\text{long}}(\mathbf{s}_j \otimes \mathbf{s}_j)]_{q_j} = [e_1 \cdot e_2]_{q_j}$. Thus, if the noises of \mathbf{c}_1 and \mathbf{c}_2 have length at most B , then the noise of \mathbf{c}_3 has length at most $\gamma_R \cdot B^2$, where γ_R is the expansion factor of R . If this length is less than $q_j/2$, then decryption works correctly. In particular, if $m_i = [\langle \mathbf{c}_i, \mathbf{s}_j \rangle]_{q_j} \bmod 2 = [e_i]_2$ for $i \in \{1, 2\}$, then over R_2 we have $[\langle \mathbf{c}_3, \mathbf{s}_j \otimes \mathbf{s}_j \rangle]_{q_j} \bmod 2 = [[e_1 \cdot e_2]_{q_j}]_2 = [e_1 \cdot e_2]_2 = [e_1]_2 \cdot [e_2]_2 = m_1 \cdot m_2$. That is, correctness is preserved as long as this noise does not wrap modulo q_j .

The correctness of FHE.Add and FHE.Mult (before FHE.Refresh) is formally captured in the following lemmas.

Lemma 6. *Let \mathbf{c}_1 and \mathbf{c}_2 be two ciphertexts under key \mathbf{s}_j for modulus q_j , where $\|[\langle \mathbf{c}_i, \mathbf{s}_j \rangle]_{q_j}\| \leq B$ and $m_i = [[\langle \mathbf{c}_i, \mathbf{s}_j \rangle]_{q_j}]_2$. Let $\mathbf{s}'_j = \mathbf{s}_j \otimes \mathbf{s}_j$, where the “non-quadratic coefficients” of \mathbf{s}'_j (namely, the ‘1’ and the coefficients of \mathbf{s}_j) are placed first. Let $\mathbf{c}' = \mathbf{c}_1 + \mathbf{c}_2$, and pad \mathbf{c}' with zeros to get a vector \mathbf{c}_3 such that $\langle \mathbf{c}_3, \mathbf{s}'_j \rangle = \langle \mathbf{c}', \mathbf{s}_j \rangle$. The noise $[\langle \mathbf{c}_3, \mathbf{s}'_j \rangle]_{q_j}$ has length at most $2B$. If $2B < q_j/2$, \mathbf{c}_3 is an encryption of $m_1 + m_2$ under key \mathbf{s}'_j for modulus q_j – i.e., $m_1 \cdot m_2 = [[\langle \mathbf{c}_3, \mathbf{s}'_j \rangle]_{q_j}]_2$.*

Lemma 7. *Let \mathbf{c}_1 and \mathbf{c}_2 be two ciphertexts under key \mathbf{s}_j for modulus q_j , where $\|[\langle \mathbf{c}_i, \mathbf{s}_j \rangle]_{q_j}\| \leq B$ and $m_i = [[\langle \mathbf{c}_i, \mathbf{s}_j \rangle]_{q_j}]_2$. Let the linear equation $L_{\mathbf{c}_1, \mathbf{c}_2}^{\text{long}}(\mathbf{x} \otimes \mathbf{x})$ be as defined above, let \mathbf{c}_3 be the coefficient vector of this linear equation, and let $\mathbf{s}'_j = \mathbf{s}_j \otimes \mathbf{s}_j$. The noise $[\langle \mathbf{c}_3, \mathbf{s}'_j \rangle]_{q_j}$ has length at most $\gamma_R \cdot B^2$. If $\gamma_R \cdot B^2 < q_j/2$, \mathbf{c}_3 is an encryption of $m_1 \cdot m_2$ under key \mathbf{s}'_j for modulus q_j – i.e., $m_1 \cdot m_2 = [[\langle \mathbf{c}_3, \mathbf{s}'_j \rangle]_{q_j}]_2$.*

The computation needed to compute the tensored ciphertext c_3 is $\tilde{O}(dn_j^2 \log q_j)$. For the RLWE instantiation, since $n_j = 1$ and since (as we will see) $\log q_j$ depends logarithmically on the security parameter and linearly on L , the computation here is only quasi-linear in the security parameter. For the LWE instantiation, the computation is quasi-quadratic.

4.3 Correctness and Performance of FHE.Refresh

FHE.Refresh consists of three steps: Expand, Switch Moduli, and Switch Keys. We address each of these steps in turn.

Correctness and Performance of the Expand Step. The Expand step of FHE.Refresh takes as input a long ciphertext c under the long tensored key $s'_j = s_j \otimes s_j$ for modulus q_j . It simply applies the Powersof2 transformation to c to obtain c_1 . By Lemma 2, we know that

$$\langle \text{Powersof2}(c, q_j), \text{BitDecomp}(s'_j, q_j) \rangle = \langle c, s'_j \rangle \bmod q_j$$

i.e., we know that if s'_j decrypts c correctly, then s''_j decrypts c_1 correctly. The noise has not been affected at all.

If implemented naively, the computation in the Expand step is $\tilde{O}(dn_j^2 \log^2 q_j)$. The somewhat high computation is due to the fact that the expanded ciphertext is a $((n_j^{j+1}) \cdot \lceil \log q_j \rceil)$ -dimensional vector over R_q .

However, recall that s_j is drawn using the distribution χ – i.e., it has small coefficients of size basically independent of q_j . Consequently, s'_j also has small coefficients, and we can use this *a priori* knowledge in combination with an optimized version of BitDecomp to output a shorter bit decomposition of s'_j – in particular, a $((n_j^{j+1}) \cdot \lceil \log q'_j \rceil)$ -dimensional vector over R_q where $q'_j \ll q_j$ is a bound (with overwhelming probability) on the coefficients of elements output by χ . Similarly, we can use an abbreviated version of Powersof2(c, q_j). In this case, the computation is $\tilde{O}(dn_j^2 \log q_j)$.

Correctness and Performance of the Switch-Moduli Step. The Switch Moduli step takes as input a ciphertext c_1 under the secret bit-vector s''_j for the modulus q_j , and outputs the ciphertext $c_2 \leftarrow \text{Scale}(c_1, q_j, q_{j-1}, 2)$, which we claim to be a ciphertext under key s''_j for modulus q_{j-1} . Note that s''_j is a *short* secret key, since it is a bit vector in $R_2^{t_j}$ for $t_j \leq (n_j^{j+1}) \cdot \lceil \log q_j \rceil$. By Corollary 1, and using the fact that $\ell_1(s''_j) \leq \sqrt{d} \cdot t_j$, the following is true: if the noise of c_1 has length at most $B < q_j/2 - (q_j/q_{j-1}) \cdot d \cdot \gamma_R \cdot t_j$, then correctness is preserved and the noise of c_2 is bounded by $(q_{j-1}/q_j) \cdot B + d \cdot \gamma_R \cdot t_j$. Of course, the key feature of this step for our purposes is that switching moduli may *reduce* the length of the moduli when $q_{j-1} < q_j$.

We capture the correctness of the Switch-Moduli step in the following lemma.

Lemma 8. *Let c_1 be a ciphertext under the key $s''_j = \text{BitDecomp}(s_j \otimes s_j, q_j)$ such that $e_j \leftarrow [\langle c_1, s''_j \rangle]_{q_j}$ has length at most B and $m = [e_j]_2$. Let $c_2 \leftarrow \text{Scale}(c_1, q_j, q_{j-1}, 2)$, and let $e_{j-1} = [\langle c_2, s''_j \rangle]_{q_{j-1}}$. Then, e_{j-1} (the new noise) has length at most $(q_{j-1}/q_j) \cdot B + d \cdot \gamma_R \cdot (n_j^{j+1}) \cdot \lceil \log q_j \rceil$, and (assuming this noise length is less than $q_{j-1}/2$) we have $m = [e_{j-1}]_2$.*

The computation in the Switch-Moduli step is $\tilde{O}(dn_j^2 \log q_j)$, using the optimized versions of BitDecomp and Powersof2 mentioned above.

Correctness and Performance of the Switch-Key Step. Finally, in the Switch Keys step, we take as input a ciphertext c_2 under key s''_j for modulus q_{j-1} and set $c_3 \leftarrow \text{SwitchKey}(\tau_{s''_j \rightarrow s_{j-1}}, c_2, q_{j-1})$, a ciphertext under the key s_{j-1} for modulus q_{j-1} . In Lemma 3, we proved the correctness of key switching and established that the noise grows only by the additive factor 2 ($\text{BitDecomp}(c_2, q_{j-1}), e$), where $\text{BitDecomp}(c_2, q_{j-1})$ is

a (short) bit-vector and \mathbf{e} is a (short and fresh) noise vector. In particular, if the noise originally had length B , then after the Switch Keys step is has length at most $B + 2 \cdot \gamma_R \cdot \sum_{i=1}^{u_j} \|\text{BitDecomp}(\mathbf{c}_2, q_{j-1})[i]\| \cdot B_\chi \leq B + 2 \cdot \gamma_R \cdot u_j \cdot \sqrt{d} \cdot B_\chi$, where $u_j \leq \binom{n_j+1}{2} \cdot \lceil \log q_j \rceil \cdot \lceil \log q_{j-1} \rceil$ is the dimension of $\text{BitDecomp}(\mathbf{c}_2)$.

We capture the correctness of the Switch-Key step in the following lemma.

Lemma 9. *Let \mathbf{c}_2 be a ciphertext under the key $\mathbf{s}_j'' = \text{BitDecomp}(\mathbf{s}_j \otimes \mathbf{s}_j, q_j)$ for modulus q_{j-1} such that $e_1 \leftarrow [\langle \mathbf{c}_2, \mathbf{s}_j'' \rangle]_{q_{j-1}}$ has length at most B and $m = [e_1]_2$. Let $\mathbf{c}_3 \leftarrow \text{SwitchKey}(\tau_{\mathbf{s}_j'' \rightarrow \mathbf{s}_{j-1}}, \mathbf{c}_2, q_{j-1})$, and let $e_2 = [\langle \mathbf{c}_3, \mathbf{s}_{j-1} \rangle]_{q_{j-1}}$. Then, e_2 (the new noise) has length at most $B + 2 \cdot \gamma_R \cdot \binom{n_j+1}{2} \cdot \lceil \log q_j \rceil^2 \cdot \sqrt{d} \cdot B_\chi$ and (assuming this noise length is less than $q_{j-1}/2$) we have $m = [e_2]_2$.*

In terms of computation, the Switch-Key step involves multiplying the transpose of u_j -dimensional vector $\text{BitDecomp}(\mathbf{c}_2)$ with a $u_j \times (n_{j-1} + 1)$ matrix B . Assuming $n_j \geq n_{j-1}$ and $q_j \geq q_{j-1}$, and using the optimized versions of BitDecomp and Powersof2 mentioned above to reduce u_j , this computation is $\tilde{O}(dn_j^3 \log^2 q_j)$. Still this is quasi-linear in the RLWE instantiation.

4.4 Putting the Pieces Together: Parameters, Correctness, Performance

So far we have established that the scheme is correct, *assuming* that the noise does not wrap modulo q_j or q_{j-1} . Now we need to show that we can set the parameters of the scheme to ensure that such wrapping never occurs.

Our strategy for setting the parameters is to pick a “universal” bound B on the noise length, and then prove, for all j , that a valid ciphertext under key \mathbf{s}_j for modulus q_j has noise length at most B . This bound B is quite small: polynomial in λ and $\log q_L$, where q_L is the largest modulus in our ladder. It is clear that such a bound B holds for fresh ciphertexts output by FHE.Enc . (Recall the discussion from Section 3.1 where we explained that we use a noise distribution χ that is essentially independent of the modulus.) The remainder of the proof is by induction – i.e., we will show that if the bound holds for two ciphertexts $\mathbf{c}_1, \mathbf{c}_2$ at level j , our lemmas above imply that the bound also holds for the ciphertext $\mathbf{c}' \leftarrow \text{FHE.Mult}(pk, \mathbf{c}_1, \mathbf{c}_2)$ at level $j - 1$. (FHE.Mult increases the noise strictly more in the worst-case than FHE.Add for any reasonable choice of parameters.)

Specifically, after the first step of FHE.Mult (without the Refresh step), the noise has length at most $\gamma_R \cdot B^2$. Then, we apply the Scale function, after which the noise length is at most $(q_{j-1}/q_j) \cdot \gamma_R \cdot B^2 + \eta_{\text{Scale},j}$, where $\eta_{\text{Scale},j}$ is some additive term. Finally, we apply the SwitchKey function, which introduces another additive term $\eta_{\text{SwitchKey},j}$. Overall, after the entire FHE.Mult step, the noise length is at most $(q_{j-1}/q_j) \cdot \gamma_R \cdot B^2 + \eta_{\text{Scale},j} + \eta_{\text{SwitchKey},j}$. We want to choose our parameters so that this bound is at most B . Suppose we set our ladder of moduli and the bound B such that the following two properties hold:

- Property 1: $B \geq 2 \cdot (\eta_{\text{Scale},j} + \eta_{\text{SwitchKey},j})$ for all j .
- Property 2: $q_j/q_{j-1} \geq 2 \cdot B \cdot \gamma_R$ for all j .

Then we have

$$(q_{j-1}/q_j) \cdot \gamma_R \cdot B^2 + \eta_{\text{Scale},j} + \eta_{\text{SwitchKey},j} \leq \frac{1}{2 \cdot B \cdot \gamma_R} \cdot \gamma_R \cdot B^2 + \frac{1}{2} \cdot B \leq B$$

It only remains to set our ladder of moduli and B so that Properties 1 and 2 hold.

Unfortunately, there is some circularity in Properties 1 and 2: q_L depends on B , which depends on q_L , albeit only polylogarithmically. However, it is easy to see that this circularity is not fatal. As a non-optimized example to illustrate this, set $B = \lambda^a \cdot L^b$ for very large constants a and b , and set $q_j \approx 2^{(j+1) \cdot \omega(\log \lambda + \log L)}$.

If a and b are large enough, B dominates $\eta_{\text{Scale},L} + \eta_{\text{SwitchKey},L}$, which is polynomial in λ and $\log q_L$, and hence polynomial in λ and L (Property 1 is satisfied). Since q_j/q_{j-1} is super-polynomial in both λ and L , it dominates $2 \cdot B \cdot \gamma_R$ (Property 2 is satisfied). In fact, it works fine to set q_j as a modulus having $(j+1) \cdot \mu$ bits for some $\mu = \theta(\log \lambda + \log L)$ with small hidden constant.

Overall, we have that q_L , the largest modulus used in the system, is $\theta(L \cdot (\log \lambda + \log L))$ bits, and $d \cdot n_L$ must be approximately that number times λ for 2^λ security.

Theorem 3. *For some $\mu = \theta(\log \lambda + \log L)$, FHE is a correct L -leveled FHE scheme – specifically, it correctly evaluates circuits of depth L with Add and Mult gates over R_2 . The per-gate computation is $\tilde{O}(d \cdot n_L^3 \cdot \log^2 q_j) = \tilde{O}(d \cdot n_L^3 \cdot L^2)$. For the LWE case (where $d = 1$), the per-gate computation is $\tilde{O}(\lambda^3 \cdot L^5)$. For the RLWE case (where $n = 1$), the per-gate computation is $\tilde{O}(\lambda \cdot L^3)$.*

The bottom line is that we have a RLWE-based leveled FHE scheme with per-gate computation that is only *quasi-linear* in the security parameter, albeit with somewhat high dependence on the number of levels in the circuit.

Let us pause at this point to reconsider the performance of previous FHE schemes in comparison to our new scheme. Specifically, as we discussed in the Introduction, in previous SWHE schemes, the ciphertext size is at least $\tilde{O}(\lambda \cdot d^2)$, where d is the *degree* of the circuit being evaluated. One may view our new scheme as a very powerful SWHE scheme in which this dependence on *degree* has been replaced with a similar dependence on *depth*. (Recall the degree of a circuit may be exponential in its depth.) Since polynomial-size circuits have polynomial depth, which is certainly not true of *degree*, our scheme can efficiently evaluate arbitrary circuits without resorting to bootstrapping.

4.5 Security

The security of FHE follows by a standard hybrid argument from the security of E, the basic scheme described in Section 3.1. We omit the details.

5 Optimizations

Despite the fact that our new FHE scheme has per-gate computation only quasi-linear in the security parameter, we present several significant ways of optimizing it. We focus primarily on the RLWE-based scheme, since it is much more efficient.

Our first optimization is *batching*. Batching allows us to reduce the per-gate computation from quasi-linear in the security parameter to *polylogarithmic*. In more detail, we show that evaluating a function f homomorphically on $\ell = \Omega(\lambda)$ blocks of encrypted data requires only *polylogarithmically* (in terms of the security parameter λ) more computation than evaluating f on the unencrypted data. (The overhead is still polynomial in the depth L of the circuit computing f .) Batching works essentially by packing multiple plaintexts into each ciphertext.

Next, we reintroduce *bootstrapping* as an optimization rather than a necessity (Section 5.2). Bootstrapping allows us to achieve per-gate computation *quasi-quadratic* in the security parameter, *independent* of the number levels in the circuit being evaluated.

In Section 5.3, we show that *batching the bootstrapping function* is a powerful combination. With this optimization, circuits whose levels mostly have width at least λ can be evaluated homomorphically with only $\tilde{O}(\lambda)$ per-gate computation, independent of the number of levels.

Finally, Section 5.5 presents a few other miscellaneous optimizations.

5.1 Batching

Suppose we want to evaluate the same function f on ℓ blocks of encrypted data. (Or, similarly, suppose we want to evaluate the same encrypted function f on ℓ blocks of plaintext data.) Can we do this using less than

ℓ times the computation needed to evaluate f on one block of data? Can we batch?

For example, consider a keyword search function that returns ‘1’ if the keyword is present in the data and ‘0’ if it is not. The keyword search function is mostly composed of a large number of equality tests that compare the target word w to all of the different subsequences of data; this is followed up by an OR of the equality test results. All of these equality tests involve running the same w -dependent function on different blocks of data. If we could batch these equality tests, it could significantly reduce the computation needed to perform keyword search homomorphically.

If we use bootstrapping as an optimization (see Section 5.2), then obviously we will be running the decryption function homomorphically on multiple blocks of data – namely, the multiple ciphertexts that need to be refreshed. Can we batch the bootstrapping function? If we could, then we might be able to drastically reduce the average per-gate cost of bootstrapping.

Smart and Vercauteren [21] were the first to rigorously analyze batching in the context of FHE. In particular, they observed that ideal-lattice-based (and RLWE-based) ciphertexts can have many plaintext slots, associated to the factorization of the plaintext space into algebraic ideals.

When we apply batching to our new RLWE-based FHE scheme, the results are pretty amazing. Evaluating f homomorphically on $\ell = \Omega(\lambda)$ blocks of encrypted data requires only *polylogarithmically* (in terms of the security parameter λ) more computation than evaluating f on the unencrypted data. (The overhead is still polynomial in the depth L of the circuit computing f .) As we will see later, for circuits whose levels mostly have width at least λ , *batching the bootstrapping function* (i.e., batching homomorphic evaluation of the decryption function) allows us to reduce the per-gate computation of our bootstrapped scheme from $\tilde{O}(\lambda^2)$ to $\tilde{O}(\lambda)$ (independent of L).

To make the exposition a bit simpler, in our RLWE-based instantiation where $R = \mathbb{Z}[x]/(x^d + 1)$, we will not use R_2 as our plaintext space, but instead use a plaintext space R_p that is isomorphic to the direct product $R_{p_1} \times \cdots \times R_{p_d}$ of many plaintext spaces (think Chinese remaindering), so that evaluating a function *once* over R_p implicitly evaluates the function *many* times in parallel over the respective smaller plaintext spaces. The p_i ’s will be *ideals* in our ring $R = \mathbb{Z}[x]/(x^d + 1)$. (One could still use R_2 as in [21], but the number theory there is a bit more involved.)

5.1.1 Some Number Theory

Let us take a very brief tour of algebraic number theory. Suppose p is a prime number satisfying $p \equiv 1 \pmod{2d}$, and let a be a primitive $2d$ -th root of unity modulo p . Then, $x^d + 1$ factors completely into linear polynomials modulo p – in particular, $x^d + 1 = \prod_{i=1}^d (x - a_i) \pmod{p}$ where $a_i = a^{2i-1} \pmod{p}$. In some sense, the converse of the above statement is also true, and this is the essence of *reciprocity* – namely, in the ring $R = \mathbb{Z}[x]/(x^d + 1)$ the prime integer p is not actually prime, but rather it splits completely into prime ideals in R – i.e., $p = \prod_{i=1}^d \mathfrak{p}_i$. The ideal \mathfrak{p}_i equals $(p, x - a_i)$ – namely, the set of all R -elements that can be expressed as $r_1 \cdot p + r_2 \cdot (x - a_i)$ for some $r_1, r_2 \in R$. Each ideal \mathfrak{p}_i has norm p – that is, roughly speaking, a $1/p$ fraction of R -elements are in \mathfrak{p}_i , or, more formally, the p cosets $0 + \mathfrak{p}_i, \dots, (p-1) + \mathfrak{p}_i$ partition R . These ideals are relative prime, and so they behave like relative prime integers. In particular, the Chinese Remainder Theorem applies: $R_p \cong R_{p_1} \times \cdots \times R_{p_d}$.

Although the prime ideals $\{\mathfrak{p}_i\}$ are relatively prime, they are close siblings, and it is easy, in some sense, to switch from one to another. One fact that we will use (when we finally apply batching to bootstrapping) is that, for any i, j there is an automorphism $\sigma_{i \rightarrow j}$ over R that maps elements of \mathfrak{p}_i to elements of \mathfrak{p}_j . Specifically, $\sigma_{i \rightarrow j}$ works by mapping an R -element $r = r(x) = r_{d-1}x^{d-1} + \cdots + r_1x + r_0$ to $r(x^{e_{ij}}) = r_{d-1}x^{e_{ij}(d-1) \bmod 2d} + \cdots + r_1x^{e_{ij}} + r_0$ where e_{ij} is some odd number in $[1, 2d]$. Notice that this automorphism just permutes the coefficients of r and fixes the free coefficient. Notationally, we will use $\sigma_{i \rightarrow j}(\mathbf{v})$ to refer to the vector that results from applying $\sigma_{i \rightarrow j}$ coefficient-wise to \mathbf{v} .

5.1.2 How Batching Works

Deploying batching inside our scheme FHE is quite straightforward. First, we pick a prime $p = 1 \bmod 2d$ of size polynomial in the security parameter. (One should exist under the GRH.)

The next step is simply to recognize that our scheme FHE works just fine when we replace the original plaintext space R_2 with R_p . There is nothing especially magical about the number 2. In the basic scheme E described in Section 3.1, $\text{E.PublicKeyGen}(params, sk)$ is modified in the obvious way so that $\mathbf{A} \cdot \mathbf{s} = p \cdot \mathbf{e}$ rather than $2 \cdot \mathbf{e}$. (This modification induces a similar modification in SwitchKeyGen .) Decryption becomes $m = [[\langle \mathbf{c}, \mathbf{s} \rangle]_q]_p$. Homomorphic operations use mod- p gates rather than boolean gates, and it is easy (if desired) to emulate boolean gates with mod- p gates – e.g., we can compute $\text{XOR}(a, b)$ for $a, b \in \{0, 1\}^2$ using mod- p gates for any p as $a + b - 2ab$. For modulus switching, we use $\text{Scale}(\mathbf{c}_1, q_j, q_{j-1}, p)$ rather than $\text{Scale}(\mathbf{c}_1, q_j, q_{j-1}, 2)$. The larger rounding error from this new scaling procedure increases the noise slightly, but this additive noise is still polynomial in the security parameter and the number of levels, and thus is still consistent with our setting of parameters. In short, FHE can easily be adapted to work with a plaintext space R_p for p of polynomial size.

The final step is simply to recognize that, by the Chinese Remainder Theorem, evaluating an arithmetic circuit over R_p on input $\mathbf{x} \in R_p^n$ implicitly evaluates, for each i , the same arithmetic circuit over R_{p_i} on input \mathbf{x} projected down to $R_{p_i}^n$. The evaluations modulo the various prime ideals do not “mix” or interact with each other.

Theorem 4. *Let $p = 1 \bmod 2d$ be a prime of size polynomial in λ . The RLWE-based instantiation of FHE using the ring $R = \mathbb{Z}[x]/(x^d + 1)$ can be adapted to use the plaintext space $R_p = \otimes_{i=1}^d R_{p_i}$ while preserving correctness and the same asymptotic performance. For any boolean circuit f of depth L , the scheme can homomorphically evaluate f on ℓ sets of inputs with per-gate computation $\tilde{O}(\lambda \cdot L^3 / \min\{d, \ell\})$.*

When $\ell \geq \lambda$, the per-gate computation is only polylogarithmic in the security parameter (still cubic in L).

5.2 Bootstrapping as an Optimization

Bootstrapping is no longer strictly necessary to achieve leveled FHE. However, in some settings, it may have some advantages:

- **Performance:** The per-gate computation is independent of the depth of the circuit being evaluated.
- **Flexibility:** Assuming circular security, a bootstrapped scheme can perform homomorphic evaluations indefinitely without needing to specify in advance, during Setup, a bound on the number of circuit levels.
- **Memory:** Bootstrapping permits short ciphertexts – e.g., encrypted using AES – to be de-compressed to longer ciphertexts that permit homomorphic operations. Bootstrapping allows us to save memory by storing data encrypted in the compressed form – e.g., under AES.

Here, we revisit bootstrapping, viewing it as an optimization rather than a necessity. We also reconsider the scheme FHE that we described in Section 3, viewing the scheme not as an end in itself, but rather as a very powerful SWHE whose performance degrades polynomially in the *depth* of the circuit being evaluated, as opposed to previous SWHE schemes whose performance degrades polynomially in the *degree*. In particular, we analyze how efficiently it can evaluate its decryption function, as needed to bootstrap. Not surprisingly, our faster SWHE scheme can also bootstrap faster. The decryption function has only logarithmic depth and can be evaluated homomorphically in time quasi-quadratic in the security parameter (for the RLWE instantiation), giving a bootstrapped scheme with quasi-quadratic per-gate computation overall.

5.2.1 Decryption as a Circuit of Quasi-Linear Size and Logarithmic Depth

Recall that the decryption function is $m = [[\langle \mathbf{c}, \mathbf{s} \rangle]_q]_2$. Suppose that we are given the “bits” (elements in R_2) of \mathbf{s} as input, and we want to compute $[[\langle \mathbf{c}, \mathbf{s} \rangle]_q]_2$ using an arithmetic circuit that has Add and Mult gates over R_2 . (When we bootstrap, of course we are given the bits of \mathbf{s} in encrypted form.) Note that we will run the decryption function homomorphically on level-0 ciphertexts – i.e., when q is small, only polynomial in the security parameter. What is the complexity of this circuit? Most importantly for our purposes, what is its depth and size? The answer is that we can perform decryption with $\tilde{O}(\lambda)$ computation and $O(\log \lambda)$ depth. Thus, in the RLWE instantiation, we can evaluate the decryption function *homomorphically* using our new scheme with quasi-quadratic computation. (For the LWE instantiation, the bootstrapping computation is quasi-quartic.)

First, let us consider the LWE case, where \mathbf{c} and \mathbf{s} are n -dimensional integer vectors. Obviously, each product $\mathbf{c}[i] \cdot \mathbf{s}[i]$ can be written as the sum of at most $\log q$ “shifts” of $\mathbf{s}[i]$. These horizontal shifts of $\mathbf{s}[i]$ use at most $2 \log q$ columns. Thus, $\langle \mathbf{c}, \mathbf{s} \rangle$ can be written as the sum of $n \cdot \log q$ numbers, where each number has $2 \log q$ digits. As discussed in [8], we can use the three-for-two trick, which takes as input three numbers in binary (of arbitrary length) and outputs (using constant depth) two binary numbers with the same sum. Thus, with $O(\log(n \cdot \log q)) = O(\log n + \log \log q)$ depth and $O(n \log^2 q)$ computation, we obtain two numbers with the desired sum, each having $O(\log n + \log q)$ bits. We can sum the final two numbers with $O(\log \log n + \log \log q)$ depth and $O(\log n + \log q)$ computation. So far, we have used depth $O(\log n + \log \log q)$ and $O(n \log^2 q)$ computation to compute $\langle \mathbf{c}, \mathbf{s} \rangle$. Reducing this value modulo q is an operation akin to division, for which there are circuits of size $\text{polylog}(q)$ and depth $\log \log q$. Finally, reducing modulo 2 just involves dropping the most significant bits. Overall, since we are interested only in the case where $\log q = O(\log \lambda)$, we have that decryption requires $\tilde{O}(\lambda)$ computation and depth $O(\log \lambda)$.

For the RLWE case, we can use the R_2 plaintext space to emulate the simpler plaintext space \mathbb{Z}_2 . Using \mathbb{Z}_2 , the analysis is basically the same as above, except that we mention that the DFT is used to multiply elements in R .

In practice, it would be useful to tighten up this analysis by reducing the polylogarithmic factors in the computation and the constants in the depth. Most likely this could be done by evaluating decryption using symmetric polynomials [8, 9] or with a variant of the “grade-school addition” approach used in the Gentry-Halevi implementation [10].

5.2.2 Bootstrapping Lazily

Bootstrapping is rather expensive computationally. In particular, the cost of bootstrapping a ciphertext is greater than the cost of a homomorphic operation by approximately a factor of λ . This suggests the question: can we lower per-gate computation of a bootstrapped scheme by bootstrapping *lazily* – i.e., applying the refresh procedure only at a $1/L$ fraction of the circuit levels for some well-chosen L [11]? Here we show that the answer is yes. By bootstrapping lazily for $L = \theta(\log \lambda)$, we can lower the per-gate computation by a logarithmic factor.

Let us present this result somewhat abstractly. Suppose that the per-gate computation for a L -level no-bootstrapping FHE scheme is $f(\lambda, L) = \lambda^{a_1} \cdot L^{a_2}$. (We ignore logarithmic factors in f , since they will not affect the analysis, but one can imagine that they add a very small ϵ to the exponent.) Suppose that bootstrapping a ciphertext requires a c -depth circuit. Since we want to be capable of evaluation depth L *after* evaluating the c levels need to bootstrap a ciphertext, the bootstrapping procedure needs to begin with ciphertexts that can be used in a $(c + L)$ -depth circuit. Consequently, let us say that the computation needed to bootstrap a ciphertext is $g(\lambda, c + L)$ where $g(\lambda, x) = \lambda^{b_1} \cdot x^{b_2}$. The overall per-gate computation is approximately $f(\lambda, L) + g(\lambda, c + L)/L$, a quantity that we seek to minimize.

We have the following lemma.

Lemma 10. *Let $f(\lambda, L) = \lambda^{a_1} \cdot L^{a_2}$ and $g(\lambda, L) = \lambda^{b_1} \cdot L^{b_2}$ for constants $b_1 > a_1$ and $b_2 > a_2 \geq 1$. Let $h(\lambda, L) = f(\lambda, L) + g(\lambda, c + L)/L$ for $c = \theta(\log \lambda)$. Then, for fixed λ , $h(\lambda, L)$ has a minimum for $L \in [(c - 1)/(b_2 - 1), c/(b_2 - 1)]$ – i.e., at some $L = \theta(\log \lambda)$.*

Proof. Clearly $h(\lambda, L) = +\infty$ at $L = 0$, then it decreases toward a minimum, and finally it eventually increases again as L goes toward infinity. Thus, $h(\lambda, L)$ has a minimum at some positive value of L . Since $f(\lambda, L)$ is monotonically increasing (i.e., the derivative is positive), the minimum must occur where the derivative of $g(\lambda, c + L)/L$ is negative. We have

$$\begin{aligned} \frac{d}{dL} g(\lambda, c + L)/L &= g'(\lambda, c + L)/L - g(\lambda, c + L)/L^2 \\ &= b_2 \cdot \lambda^{b_1} \cdot (c + L)^{b_2-1}/L - \lambda^{b_1} \cdot (c + L)^{b_2}/L^2 \\ &= (\lambda^{b_1} \cdot (c + L)^{b_2-1}/L^2) \cdot (b_2 \cdot L - c - L), \end{aligned}$$

which becomes positive when $L \geq c/(b_2 - 1)$ – i.e., the derivative is negative only when $L = O(\log \lambda)$. For $L < (c - 1)/(b_2 - 1)$, we have that the above derivative is less than $-\lambda^{b_1} \cdot (c + L)^{b_2-1}/L^2$, which dominates the positive derivative of f . Therefore, for large enough value of λ , the value $h(\lambda, L)$ has its minimum at some $L \in [(c - 1)/(b_2 - 1), c/(b_2 - 1)]$. \square

This lemma basically says that, since homomorphic decryption takes $\theta(\log \lambda)$ levels and its cost is super-linear and dominates that of normal homomorphic operations (FHE.Add and FHE.Mult), it makes sense to bootstrap lazily – in particular, once every $\theta(\log \lambda)$ levels. (If one bootstrapped even more lazily than this, the super-linear cost of bootstrapping begins to ensure that the (amortized) per-gate cost of bootstrapping alone is increasing.) It is easy to see that, since the per-gate computation is dominated by bootstrapping, bootstrapping lazily every $\theta(\log \lambda)$ levels reduces the per-gate computation by a factor of $\theta(\log \lambda)$.

5.3 Batching the Bootstrapping Operation

Suppose that we are evaluating a circuit homomorphically, that we are currently at a level in the circuit that has at least d gates (where d is the dimension of our ring), and that we want to bootstrap (refresh) all of the ciphertexts corresponding to the respective wires at that level. That is, we want to homomorphically evaluate the decryption function at least d times in parallel. This seems like an ideal place to apply batching.

However, there are some nontrivial problems. In Section 5.1, our focus was rather limited. For example, we did not consider whether homomorphic operations could continue after the batched computation. Indeed, at first glance, it would appear that homomorphic operations *cannot* continue, since, after batching, the encrypted data is partitioned into non-interacting relatively-prime plaintext slots, whereas the whole point of homomorphic encryption is that the encrypted data can interact (within a common plaintext slot). Similarly, we did not consider homomorphic operations *before* the batched computation. Somehow, we need the input to the batched computation to come pre-partitioned into the different plaintext slots.

What we need are Pack and Unpack functions that allow the batching procedure to interface with “normal” homomorphic operations. One may think of the Pack and Unpack functions as an on-ramp to and an exit-ramp from the “fast lane” of batching. Let us say that normal homomorphic operations will always use the plaintext slot R_{p_1} . Roughly, the Pack function should take a bunch of ciphertexts $\mathbf{c}_1, \dots, \mathbf{c}_d$ that encrypt messages $m_1, \dots, m_d \in \mathbb{Z}_p$ under key \mathbf{s}_1 for modulus q and plaintext slot R_{p_1} , and then *aggregate* them into a single ciphertext \mathbf{c} under some possibly different key \mathbf{s}_2 for modulus q and plaintext slot $R_p = \otimes_{i=1}^d R_{p_i}$, so that correctness holds with respect to all of the different plaintext slots – i.e. $m_i = [[\langle \mathbf{c}, \mathbf{s}_2 \rangle]_q]_{p_i}$ for all i . The Pack function thus allows normal homomorphic operations to feed into the batch operation.

The Unpack function should accept the output of a batched computation, namely a ciphertext \mathbf{c}' such that $m_i = [[\langle \mathbf{c}', \mathbf{s}'_1 \rangle]_q]_{\mathbf{p}_i}$ for all i , and then de-aggregate this ciphertext by outputting ciphertexts $\mathbf{c}'_1, \dots, \mathbf{c}'_d$ under some possibly different common secret key \mathbf{s}'_2 such that $m_i = [[\langle \mathbf{c}'_i, \mathbf{s}'_2 \rangle]_q]_{\mathbf{p}_1}$ for all i . Now that all of the ciphertexts are under a common key and plaintext slot, normal homomorphic operations can resume. With such Pack and Unpack functions, we could indeed batch the bootstrapping operation. For circuits of large width (say, at least d) we could reduce the per-gate bootstrapping computation by a factor of d , making it only quasi-linear in λ . Assuming the Pack and Unpack functions have complexity at most quasi-quadratic in d (per-gate this is only quasi-linear, since Pack and Unpack operate on d gates), the overall per-gate computation of a batched-bootstrapped scheme becomes only quasi-linear.

Here, we describe suitable Pack and Unpack functions. These functions will make heavy use of the automorphisms $\sigma_{i \rightarrow j}$ over R that map elements of \mathbf{p}_i to elements of \mathbf{p}_j . (See Section 5.1.1.) We note that Smart and Vercauteren [21] used these automorphisms to construct something similar to our Pack function (though for unpacking they resorted to bootstrapping). We also note that Lyubashevsky, Peikert and Regev [14] used these automorphisms to permute the ideal factors \mathbf{q}_i of the modulus q , which was an essential tool toward their proof of the pseudorandomness of RLWE.

Toward Pack and Unpack procedures, our main idea is the following. If m is encoded in the free term as a number in $\{0, \dots, p-1\}$ and if $m = [[\langle \mathbf{c}, \mathbf{s} \rangle]_q]_{\mathbf{p}_i}$, then $m = [[\langle \sigma_{i \rightarrow j}(\mathbf{c}), \sigma_{i \rightarrow j}(\mathbf{s}) \rangle]_q]_{\mathbf{p}_j}$. That is, we can switch the plaintext slot but leave the decrypted message unchanged by applying the same automorphism to the ciphertext and the secret key. (These facts follow from the fact that $\sigma_{i \rightarrow j}$ is a homomorphism, that it maps elements of \mathbf{p}_i to elements of \mathbf{p}_j , and that it fixes free terms.) Of course, then we have a problem: the ciphertext is now under a different key, whereas we may want the ciphertext to be under the same key as other ciphertexts. To get the ciphertexts to be back under the same key, we simply use the SwitchKey algorithm to switch all of the ciphertexts to a new common key.

Some technical remarks before we describe Pack and Unpack more formally: We mention again that E.PublicKeyGen is modified in the obvious way so that $\mathbf{A} \cdot \mathbf{s} = p \cdot \mathbf{e}$ rather than $2 \cdot \mathbf{e}$, and that this modification induces a similar modification in SwitchKeyGen. Also, let $u \in R$ be a short element such that $u \in 1 + \mathbf{p}_1$ and $u \in \mathbf{p}_j$ for all $j \neq 1$. It is obvious that such a u with coefficients in $(-p/2, p/2]$ can be computed efficiently by first picking *any* element u' such that $u' \in 1 + \mathbf{p}_1$ and $u' \in \mathbf{p}_j$ for all $j \neq 1$, and then reducing the coefficients of u' modulo p .

PackSetup($\mathbf{s}_1, \mathbf{s}_2$): Takes as input two secret keys $\mathbf{s}_1, \mathbf{s}_2$. For all $i \in [1, d]$, it runs $\tau_{\sigma_{1 \rightarrow i}(\mathbf{s}_1) \rightarrow \mathbf{s}_2} \leftarrow \text{SwitchKeyGen}(\sigma_{1 \rightarrow i}(\mathbf{s}_1), \mathbf{s}_2)$.

Pack($\{\mathbf{c}_i\}_{i=1}^d, \{\tau_{\sigma_{1 \rightarrow i}(\mathbf{s}_1) \rightarrow \mathbf{s}_2}\}_{i=1}^d$): Takes as input ciphertexts $\mathbf{c}_1, \dots, \mathbf{c}_d$ such that $m_i = [[\langle \mathbf{c}_i, \mathbf{s}_1 \rangle]_q]_{\mathbf{p}_1}$ and $0 = [[\langle \mathbf{c}_i, \mathbf{s}_1 \rangle]_q]_{\mathbf{p}_j}$ for all $j \neq 1$, and also some auxiliary information output by PackSetup. For all i , it does the following:

- Computes $\mathbf{c}_i^* \leftarrow \sigma_{1 \rightarrow i}(\mathbf{c}_i)$. (Observe: $m_i = [[\langle \mathbf{c}_i^*, \sigma_{1 \rightarrow i}(\mathbf{s}_1) \rangle]_q]_{\mathbf{p}_i}$ while $0 = [[\langle \mathbf{c}_i^*, \sigma_{1 \rightarrow i}(\mathbf{s}_1) \rangle]_q]_{\mathbf{p}_j}$ for all $j \neq i$.)
- Runs $\mathbf{c}_i^\dagger \leftarrow \text{SwitchKey}(\tau_{\sigma_{1 \rightarrow i}(\mathbf{s}_1) \rightarrow \mathbf{s}_2}, \mathbf{c}_i^*)$ (Observe: Assuming the noise does not wrap, we have that $m_i = [[\langle \mathbf{c}_i^\dagger, \mathbf{s}_2 \rangle]_q]_{\mathbf{p}_i}$ and $0 = [[\langle \mathbf{c}_i^\dagger, \mathbf{s}_2 \rangle]_q]_{\mathbf{p}_j}$ for all $j \neq i$.)

Finally, it outputs $\mathbf{c} \leftarrow \sum_{i=1}^d \mathbf{c}_i^\dagger$. (Observe: Assuming the noise does not wrap, we have that $m_i = [[\langle \mathbf{c}, \mathbf{s}_2 \rangle]_q]_{\mathbf{p}_i}$ for all i .)

UnpackSetup($\mathbf{s}_1, \mathbf{s}_2$): Takes as input two secret keys $\mathbf{s}_1, \mathbf{s}_2$. For all $i \in [1, d]$, it runs $\tau_{\sigma_{i \rightarrow 1}(\mathbf{s}_1) \rightarrow \mathbf{s}_2} \leftarrow \text{SwitchKeyGen}(\sigma_{i \rightarrow 1}(\mathbf{s}_1), \mathbf{s}_2)$.

$\text{Unpack}(\mathbf{c}, \{\tau_{\sigma_{i \rightarrow 1}(\mathbf{s}_1) \rightarrow \mathbf{s}_2}\}_{i=1}^d)$: Takes as input a ciphertext \mathbf{c} such that $m_i = [[\langle \mathbf{c}, \mathbf{s}_1 \rangle]_q]_{p_i}$ for all i , and also some auxiliary information output by UnpackSetup . For all i , it does the following:

- Computes $\mathbf{c}_i \leftarrow u \cdot \sigma_{i \rightarrow 1}(\mathbf{c})$. (Observe: Assuming the noise does not wrap, $m_i = [[\langle \mathbf{c}_i, \sigma_{i \rightarrow 1}(\mathbf{s}_1) \rangle]_q]_{p_1}$ and $0 = [[\langle \mathbf{c}_i, \sigma_{i \rightarrow 1}(\mathbf{s}_1) \rangle]_q]_{p_j}$ for all $j \neq 1$.)
- Outputs $\mathbf{c}_i^* \leftarrow \text{SwitchKey}(\tau_{\sigma_{i \rightarrow 1}(\mathbf{s}_1) \rightarrow \mathbf{s}_2}, \mathbf{c}_i)$. (Observe: Assuming the noise does not wrap, $m_i = [[\langle \mathbf{c}_i^*, \mathbf{s}_2 \rangle]_q]_{p_1}$ and $0 = [[\langle \mathbf{c}_i^*, \mathbf{s}_2 \rangle]_q]_{p_j}$ for all $j \neq 1$.)

Splicing the Pack and Unpack procedures into our scheme FHE is tedious but pretty straightforward. Although these procedures introduce many more encrypted secret keys, this does not cause a circular security problem as long as the chain of encrypted secret keys is acyclic; then the standard hybrid argument applies. After applying Pack or Unpack, one may apply modulus reduction to reduce the noise back down to normal.

5.4 More Fun with Funky Plaintext Spaces

In some cases, it might be nice to have a plaintext space isomorphic to \mathbb{Z}_p for some *large* prime p – e.g., one *exponential* in the security parameter. So far, we have been using R_p as our plaintext space, and (due to the rounding step in modulus switching) the size of the noise after modulus switching is proportional to p . When p is exponential, our previous approach for handling the noise (which keeps the magnitude of the noise polynomial in λ) obviously breaks down.

To get a plaintext space isomorphic to \mathbb{Z}_p that works for exponential p , we need a new approach. Instead of using an integer modulus, we will use an *ideal* modulus I (an ideal of R) whose *norm* is some large prime p , but such that we have a basis B_I of I that is very short – e.g. $\|B_I\| = O(\text{poly}(d) \cdot p^{1/d})$. Using an ideal plaintext space forces us to modify the modulus switching technique nontrivially.

Originally, when our plaintext space was R_2 , each of the moduli in our “ladder” was odd – that is, they were all congruent to each other modulo 2 and relatively prime to 2. Similarly, we will have to choose each of the moduli in our new ladder so that they are all congruent to each other modulo I . (This just seems necessary to get the scaling to work, as the reader will see shortly.) This presents a difficulty, since we wanted the norm of I to be large – e.g., exponential in the security parameter. If we choose our moduli q_j to be *integers*, then we have that the integer $q_{j+1} - q_j \in I$ – in particular, $q_{j+1} - q_j$ is a multiple of I ’s norm, implying that the q_j ’s are exponential in the security parameter. Having such large q_j ’s does not work well in our scheme, since the underlying lattice problems becomes easy when q_j/B is exponential in d where B is a bound of the noise distribution of fresh ciphertexts, and since we need B to remain quite small for our new noise management approach to work effectively. So, instead, our ladder of moduli will also consist of ideals – in particular, principle ideals (q_j) generated by an element of $q_j \in R$. Specifically, it is easy to generate a ladder of q_j ’s that are all congruent to 1 moduli I by sampling appropriately-sized elements q_j of the coset $1 + I$ (using our short basis of I), and testing whether the principal ideal (q_j) generated by the element has appropriate norm.

Now, let us reconsider modulus switching in light of the fact that our moduli are now principal ideals. We need an analogue of Lemma 4 that works for ideal moduli.

Let us build up some notation and concepts that we will need in our new lemma. Let \mathcal{P}_q be the half-open *parallelepiped* associated to the *rotation basis* of $q \in R$. The rotation basis \mathbf{B}_q of q is the d -dimensional basis formed by the coefficient vectors of the polynomials $x^i q(x) \bmod f(x)$ for $i \in [0, d-1]$. The associated parallelepiped is $\mathcal{P}_q = \{\sum z_i \cdot \mathbf{b}_i : \mathbf{b}_i \in \mathbf{B}_q, z_i \in [-1/2, 1/2)\}$. We need two concepts associated to this parallelepiped. First, we will still use the notation $[a]_q$, but where q is now an R -element rather than integer. This notation refers to a reduced modulo the *rotation basis* of a – i.e., the element $[a]_q$ such that $[a]_q - a \in qR$ and $[a]_q \in \mathcal{P}_q$. Next, we need notions of the *inner radius* $r_{q,in}$ and *outer radius* $r_{q,out}$ of \mathcal{P}_q – that is, the

largest radius of a ball that is circumscribed by \mathcal{P}_q , and the smallest radius of a ball that circumscribes \mathcal{P}_q . It is possible to choose q so that the ratio $r_{q,out}/r_{q,in}$ is $\text{poly}(d)$. For example, this is true when q is an integer. For a suitable value of $f(x)$ that determines our ring, such as $f(x) = x^d + 1$, the expected value of ratio will be $\text{poly}(d)$ even if q is sampled uniformly (e.g., according to discrete Gaussian distribution centered at 0). More generally, we will refer to $r_{\mathbf{B},out}$ as the outer radius associated to the parallelepiped determined by basis \mathbf{B} . Also, in the field $\mathbb{Q}(x)/f(x)$ overlying this ring, it will be true with overwhelming probability, if q is sampled uniformly, that $\|q^{-1}\| = 1/\|q\|$ up to a $\text{poly}(d)$ factor. For convenience, let $\alpha(d)$ be a polynomial such that $\|q^{-1}\| = 1/\|q\|$ up to a $\alpha(d)$ factor and moreover $r_{q,out}/r_{q,in}$ is at most $\alpha(d)$ with overwhelming probability. For such an α , we say q is α -good. Finally, in the lemma, γ_R denotes the expansion factor of R – i.e., $\max\{\|\mathbf{a} \cdot \mathbf{b}\|/\|\mathbf{a}\|\|\mathbf{b}\| : \mathbf{a}, \mathbf{b} \in R\}$.

Lemma 11. *Let q_1 and q_2 , $\|q_1\| < \|q_2\|$, be two α -good elements of R . Let \mathbf{B}_I be a short basis (with outer radius $r_{\mathbf{B}_I,out}$) of an ideal I of R such that $q_1 - q_2 \in I$. Let \mathbf{c} be an integer vector and $\mathbf{c}' \leftarrow \text{Scale}(\mathbf{c}, q_2, q_1, I)$ – that is, \mathbf{c}' is an R -element at most $2r_{\mathbf{B}_I,out}$ distant from $(q_1/q_2) \cdot \mathbf{c}$ such that $\mathbf{c}' - \mathbf{c} \in I$. Then, for any \mathbf{s} with*

$$\|[\langle \mathbf{c}, \mathbf{s} \rangle]_{q_2}\| < \left(r_{q_2,in}/\alpha(d)^2 - (\|q_2\|/\|q_1\|)\gamma_R \cdot 2r_{\mathbf{B}_I,out} \cdot \ell_1^{(R)}(\mathbf{s}) \right) / (\alpha(d) \cdot \gamma_R^2)$$

we have

$$[\langle \mathbf{c}', \mathbf{s} \rangle]_{q_1} = [\langle \mathbf{c}, \mathbf{s} \rangle]_{q_2} \bmod I \quad \text{and} \quad \|[\langle \mathbf{c}', \mathbf{s} \rangle]_{q_1}\| < \alpha(d) \cdot \gamma_R^2 \cdot (\|q_1\|/\|q_2\|) \cdot \|[\langle \mathbf{c}, \mathbf{s} \rangle]_{q_2}\| + \gamma_R \cdot 2r_{\mathbf{B}_I,out} \cdot \ell_1^{(R)}(\mathbf{s})$$

where $\ell_1^{(R)}(\mathbf{s})$ is defined as $\sum_i \|\mathbf{s}[i]\|$.

Proof. We have

$$[\langle \mathbf{c}, \mathbf{s} \rangle]_{q_2} = \langle \mathbf{c}, \mathbf{s} \rangle - kq_2$$

for some $k \in R$. For the same k , let

$$e_{q_1} = \langle \mathbf{c}', \mathbf{s} \rangle - kq_1 \in R$$

Note that $e_{q_1} = [\langle \mathbf{c}', \mathbf{s} \rangle]_{q_1} \bmod q_1$. We claim that $\|e_{q_1}\|$ is so small that $e_{q_1} = [\langle \mathbf{c}', \mathbf{s} \rangle]_{q_1}$. We have:

$$\begin{aligned} \|e_{q_1}\| &= \| -kq_1 + \langle (q_1/q_2) \cdot \mathbf{c}, \mathbf{s} \rangle + \langle \mathbf{c}' - (q_1/q_2) \cdot \mathbf{c}, \mathbf{s} \rangle \| \\ &\leq \| -kq_1 + \langle (q_1/q_2) \cdot \mathbf{c}, \mathbf{s} \rangle \| + \| \langle \mathbf{c}' - (q_1/q_2) \cdot \mathbf{c}, \mathbf{s} \rangle \| \\ &\leq \gamma_R \cdot \|q_1/q_2\| \cdot \|[\langle \mathbf{c}, \mathbf{s} \rangle]_{q_2}\| + \gamma_R \cdot 2r_{\mathbf{B}_I,out} \cdot \ell_1^{(R)}(\mathbf{s}) \\ &\leq \gamma_R^2 \cdot \|q_1\| \cdot \|q_2^{-1}\| \cdot \|[\langle \mathbf{c}, \mathbf{s} \rangle]_{q_2}\| + \gamma_R \cdot 2r_{\mathbf{B}_I,out} \cdot \ell_1^{(R)}(\mathbf{s}) \\ &\leq \alpha(d) \cdot \gamma_R^2 \cdot (\|q_1\|/\|q_2\|) \cdot \|[\langle \mathbf{c}, \mathbf{s} \rangle]_{q_2}\| + \gamma_R \cdot 2r_{\mathbf{B}_I,out} \cdot \ell_1^{(R)}(\mathbf{s}) \end{aligned}$$

By the final expression above, we see that the magnitude of e_{q_1} may actually be less than the magnitude of e_{q_2} if $\|q_1\|/\|q_2\|$ is small enough. Let us continue with the inequalities, substituting in the bound for $\|[\langle \mathbf{c}, \mathbf{s} \rangle]_{q_2}\|$:

$$\begin{aligned} \|e_{q_1}\| &\leq \alpha(d) \cdot \gamma_R^2 \cdot (\|q_1\|/\|q_2\|) \cdot \left(r_{q_2,in}/\alpha(d)^2 - (\|q_2\|/\|q_1\|)\gamma_R \cdot 2r_{\mathbf{B}_I,out} \cdot \ell_1^{(R)}(\mathbf{s}) \right) / (\alpha(d) \cdot \gamma_R^2) \\ &\quad + \gamma_R \cdot 2r_{\mathbf{B}_I,out} \cdot \ell_1^{(R)}(\mathbf{s}) \\ &\leq (\|q_1\|/\|q_2\|) \cdot \left(r_{q_2,in}/\alpha(d)^2 - (\|q_2\|/\|q_1\|)\gamma_R \cdot 2r_{\mathbf{B}_I,out} \cdot \ell_1^{(R)}(\mathbf{s}) \right) + \gamma_R \cdot 2r_{\mathbf{B}_I,out} \cdot \ell_1^{(R)}(\mathbf{s}) \\ &\leq \left(r_{q_1,in} - \gamma_R \cdot 2r_{\mathbf{B}_I,out} \cdot \ell_1^{(R)}(\mathbf{s}) \right) + \gamma_R \cdot 2r_{\mathbf{B}_I,out} \cdot \ell_1^{(R)}(\mathbf{s}) \\ &= r_{q_1,in} \end{aligned}$$

Since $\|e_{q_1}\| < r_{q_1, \text{in}}$, e_{q_1} is inside the parallelepiped \mathcal{P}_{q_1} and it is indeed true that $e_{q_1} = [\langle \mathbf{c}', \mathbf{s} \rangle]_{q_1}$. Furthermore, we have $[\langle \mathbf{c}', \mathbf{s} \rangle]_{q_1} = e_{q_1} = \langle \mathbf{c}', \mathbf{s} \rangle - kq_1 = \langle \mathbf{c}, \mathbf{s} \rangle - kq_2 = [\langle \mathbf{c}, \mathbf{s} \rangle]_{q_2} \bmod I$. \square

The bottom line is that we can apply the modulus switching technique to moduli that are ideals, and this allows us to use, if desired, plaintext spaces that are very large (exponential in the security parameter) and that have properties that are often desirable (such as being isomorphic to a large prime field).

5.5 Other Optimizations

If one is willing to assume circular security, the keys $\{\mathbf{s}_j\}$ may all be the same, thereby permitting a public key of size independent of L .

While it is not necessary, squashing may still be a useful optimization in practice, as it can be used to lower the depth of the decryption function, thereby reducing the size of the largest modulus needed in the scheme, which may improve efficiency.

For the LWE-based scheme, one can use key switching to gradually reduce the dimension n_j of the ciphertext (and secret key) vectors as q_j decreases – that is, as one traverses to higher levels in the circuit. As q_j decreases, the associated LWE problem becomes (we believe) progressively harder (for a fixed noise distribution χ). This allows one to gradually reduce the dimension n_j without sacrificing security, and reduce ciphertext length faster (as one goes higher in the circuit) than one could simply by decreasing q_j alone.

6 Summary and Future Directions

Our RLWE-based FHE scheme without bootstrapping requires only $\tilde{O}(\lambda \cdot L^3)$ per-gate computation where L is the depth of the circuit being evaluated, while the bootstrapped version has only $\tilde{O}(\lambda^2)$ per-gate computation. For circuits of width $\Omega(\lambda)$, we can use batching to reduce the per-gate computation of the bootstrapped version by another factor of λ .

While these schemes should perform significantly better than previous FHE schemes, we caution that the polylogarithmic factors in the per-gate computation are large. One future direction toward a truly practical scheme is to tighten up these polylogarithmic factors considerably.

Acknowledgments. We thank Carlos Aguilar Melchor, Boaz Barak, Shai Halevi, Chris Peikert, Nigel Smart, and Jiang Zhang for helpful discussions and insights.

References

- [1] Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In *CRYPTO*, volume 5677 of *Lecture Notes in Computer Science*, pages 595–618. Springer, 2009.
- [2] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-DNF formulas on ciphertexts. In *Proceedings of Theory of Cryptography Conference 2005*, volume 3378 of *LNCS*, pages 325–342, 2005.
- [3] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. Manuscript, to appear in FOCS 2011, available at <http://eprint.iacr.org/2011/344>.
- [4] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. Manuscript, to appear in CRYPTO 2011.
- [5] Jean-Sébastien Coron, Avradip Mandal, David Naccache, and Mehdi Tibouchi. Fully-homomorphic encryption over the integers with shorter public-keys. Manuscript, to appear in Crypto 2011.

- [6] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *Advances in Cryptology - EUROCRYPT'10*, volume 6110 of *Lecture Notes in Computer Science*, pages 24–43. Springer, 2010. Full version available on-line from <http://eprint.iacr.org/2009/616>.
- [7] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. crypto.stanford.edu/craig.
- [8] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *STOC*, pages 169–178. ACM, 2009.
- [9] Craig Gentry and Shai Halevi. Fully homomorphic encryption without squashing using depth-3 arithmetic circuits. Manuscript, to appear in FOCS 2011, available at <http://eprint.iacr.org/2011/279>.
- [10] Craig Gentry and Shai Halevi. Implementing gentry’s fully-homomorphic encryption scheme. In *EUROCRYPT*, volume 6632 of *Lecture Notes in Computer Science*, pages 129–148. Springer, 2011.
- [11] Shai Halevi, 2011. Personal communication.
- [12] Yuval Ishai and Anat Paskin. Evaluating branching programs on encrypted data. In Salil P. Vadhan, editor, *TCC*, volume 4392 of *Lecture Notes in Computer Science*, pages 575–594. Springer, 2007.
- [13] Kristin Lauter, Michael Naehrig, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? Manuscript at <http://eprint.iacr.org/2011/405>, 2011.
- [14] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *EUROCRYPT*, volume 6110 of *Lecture Notes in Computer Science*, pages 1–23, 2010.
- [15] Carlos Aguilar Melchor, Philippe Gaborit, and Javier Herranz. Additively homomorphic encryption with -operand multiplications. In Tal Rabin, editor, *CRYPTO*, volume 6223 of *Lecture Notes in Computer Science*, pages 138–154. Springer, 2010.
- [16] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In *STOC*, pages 333–342. ACM, 2009.
- [17] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *STOC*, pages 84–93. ACM, 2005.
- [18] Oded Regev. The learning with errors problem (invited survey). In *IEEE Conference on Computational Complexity*, pages 191–204. IEEE Computer Society, 2010.
- [19] Ron Rivest, Leonard Adleman, and Michael L. Dertouzos. On data banks and privacy homomorphisms. In *Foundations of Secure Computation*, pages 169–180, 1978.
- [20] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In *Public Key Cryptography - PKC'10*, volume 6056 of *Lecture Notes in Computer Science*, pages 420–443. Springer, 2010.
- [21] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic SIMD operations. Manuscript at <http://eprint.iacr.org/2011/133>, 2011.

- [22] Damien Stehlé and Ron Steinfeld. Faster fully homomorphic encryption. In *ASIACRYPT*, volume 6477 of *Lecture Notes in Computer Science*, pages 377–394. Springer, 2010.

Targeted Malleability: Homomorphic Encryption for Restricted Computations

Dan Boneh* Gil Segev† Brent Waters‡

Abstract

We put forward the notion of *targeted malleability*: given a homomorphic encryption scheme, in various scenarios we would like to restrict the homomorphic computations one can perform on encrypted data. We introduce a precise framework, generalizing the foundational notion of *non-malleability* introduced by Dolev, Dwork, and Naor (SICOMP '00), ensuring that the malleability of a scheme is targeted only at a specific set of “allowable” functions.

In this setting we are mainly interested in the efficiency of such schemes as a function of the number of repeated homomorphic operations. Whereas constructing a scheme whose ciphertext grows linearly with the number of such operations is straightforward, obtaining more realistic (or merely non-trivial) length guarantees is significantly more challenging.

We present two constructions that transform any homomorphic encryption scheme into one that offers targeted malleability. Our constructions rely on standard cryptographic tools and on succinct non-interactive arguments, which are currently known to exist in the standard model based on variants of the knowledge-of-exponent assumption. The two constructions offer somewhat different efficiency guarantees, each of which may be preferable depending on the underlying building blocks.

Keywords: Homomorphic encryption, Non-malleable encryption.

*Stanford University. Supported by NSF, DARPA, and AFOSR.

†Microsoft Research, Mountain View, CA 94043, USA.

‡University of Texas at Austin. Supported by NSF CNS-0716199, CNS-0915361, and CNS-0952692, DARPA PROCEED, Air Force Office of Scientific Research (AFO SR) MURI, DHS Grant 2006-CS-001-000001-02, and the Sloan Foundation.

1 Introduction

Fully homomorphic encryption [RAD78, Gen09, SV10, vDGH⁺10] is a remarkable development in cryptography enabling anyone to compute arbitrary functions on encrypted data. In many settings, however, the data owner may wish to restrict the class of homomorphic computations to a certain set \mathcal{F} of allowable functions. In this paper we put forward the notion of “targeted malleability”: given an encryption scheme that supports homomorphic operations with respect to some set of functions \mathcal{F} , we would like to ensure that the malleability of the scheme is targeted only at the set \mathcal{F} . That is, it should not be possible to apply any homomorphic operation other than the ones in \mathcal{F} .

Enforcing targeted malleability can be simply done by requiring the entity performing the homomorphic operation to embed a proof in the ciphertext showing that the ciphertext was computed using an allowable function. The decryptor then verifies the proof before decrypting the ciphertext, and outputs \perp if the proof is invalid. Unfortunately, as the homomorphic operation is repeated the number of proofs grows making the ciphertext grow at least *linearly* with the number of repeated homomorphic operations. It is not difficult to see that targeted malleability with a linear-size ciphertext is trivial to construct: Use any non-malleable encryption scheme, and embed in the ciphertext a description of all the functions being computed. The decryptor decrypts the original ciphertext and applies the embedded functions to it (verifying, of course, that these functions are in the allowable set).

Minimizing ciphertext expansion. Targeted malleability is much harder to construct once we require that ciphertext growth is at most sub-linear in the number of repeated homomorphic operations. Our goal is to construct systems where even after t applications of the homomorphic operation the ciphertext length does not increase much. In our main construction we are able to completely shift the dependence on t from the ciphertext to the public key: the ciphertext size is essentially independent of t . This is a natural goal since public keys are typically much more static than ciphertexts which are frequently generated and transmitted.

Motivation. While targeted malleability is an interesting concept in its own right, it has many applications in cryptography and beyond. We give a few illustrative examples:

- A spam filter implemented in a mail server adds a **spam** tag to encrypted emails whose content satisfies a certain spam predicate. The filter should be allowed to run the spam predicate, but should not modify the email contents. In this case, the set of allowable functions \mathcal{F} would be the set of allowable spam predicates and nothing else. As email passes from one server to the next each server homomorphically computes its spam predicate on the encrypted output of the previous server. Each spam filter in the chain can run its chosen spam predicate and nothing else.
- More generally, in a distributed system users initiate encrypted requests to various servers. To service a request a server may need to contact another server and that server may need to contact another, resulting in a chain of messages from server to server until the transaction is fulfilled. Each server along the way has an allowed set of operations it can apply to a received message and it should be unable to apply any operation outside this approved set.
- In a voting system based on homomorphic encryption (e.g. [CGS97]) voters take turns incrementing an encrypted vote tally using a homomorphic operation. They are only allowed to increase the encrypted tally by 1 (indicating a vote for the candidate) or by 0 (indicating a

no vote for the candidate). In elections where each voter votes for one of ℓ candidates, voters modify the encrypted tallies by adding an ℓ -bit vector, where exactly one entry is 1 and the rest are all 0's. They should be unable to modify the counters in any other way.

In all these examples there is a need to repeatedly apply a restricted homomorphic operation on encrypted data. Limiting ciphertext expansion is highly desirable.

1.1 Our Contributions

We begin by introducing a precise framework for modeling targeted malleability. Our notion of security generalizes the foundational one of non-malleability due to Dolev, Dwork, and Naor [DDN00], and is also inspired by the refinements of Bellare and Sahai [BS99], and Pass, Shelat, and Vaikuntanathan [PSV07]. Given a public-key encryption scheme that is homomorphic with respect to a set of functions \mathcal{F} we would like to capture the following intuitive notion of security¹: For any efficient adversary that is given an encryption c of a message m and outputs an encryption c' of a message m' , it should hold that either (1) m' is independent of m , (2) $c' = c$ (and thus $m' = m$), or (3) c' is obtained by repeatedly applying the homomorphic evaluation algorithm on c using functions $f_1, \dots, f_\ell \in \mathcal{F}$. The first two properties are the standard ones for non-malleable encryption, and the third property captures our new notion of targeted malleability: we would like to target the malleability of the scheme only at the class \mathcal{F} (we note that by setting $\mathcal{F} = \emptyset$ we recover the standard definition of non-malleability)². We consider this notion of security with respect to both chosen-plaintext attacks (CPA) and a-priori chosen-ciphertext attacks (CCA1)³.

We emphasize that we do not make the assumption that the set of functions \mathcal{F} is closed under composition. In particular, our approach is sufficiently general to allow targeting the malleability of a scheme at any *subset* $\mathcal{F}' \subseteq \mathcal{F}$ of the homomorphic operations that are supported by the scheme. This is significant, for example, when dealing with fully homomorphic schemes, where any set of functions is in fact a subset of the supported homomorphic operations (see Section 1.3 for more details).

Next, we present two general transformations that transform any homomorphic encryption scheme into one that enjoys targeted malleability for a limited number of repeated homomorphic operations. The resulting schemes are secure even in the setting of a-priori chosen-ciphertext attacks (CCA1). The two constructions offer rather different trade-offs in terms of efficiency. In this overview we focus on our first construction, as it already captures the main ideas underlying our methodology.

1.2 Overview of Our Approach

Our approach is based on bridging between two seemingly conflicting goals: on one hand, we would like to turn the underlying homomorphic scheme into a somewhat non-malleable one, whereas on the other hand we would like to preserve its homomorphic properties. We demonstrate that the Naor-Yung “double encryption” paradigm for non-malleability [NY90, DDN00, Sah99, Lin06] can be utilized to obtain an interesting balance between these two goals. The structure of ciphertexts in our construction follows the latter paradigm: a ciphertext is a 3-tuple (c_0, c_1, π) containing two

¹For simplicity we focus here on univariate functions and refer the reader to Section 3 for the more general case of multivariate functions

²We assume in this informal discussion that the adversary outputs a *valid* ciphertext, but our notion of security in fact considers the more general case – see Section 3.

³See Section 6 for a discussion on a-posteriori chosen-ciphertext attacks (CCA2) in the setting of homomorphic encryption, following the work of Prabhakaran and Rosulek [PR08].

encryptions of the same message using the underlying encryption scheme under two different keys along with a proof π that the ciphertext is well formed. For ciphertexts that are produced by the encryption algorithm, π is a non-interactive zero-knowledge proof, and for ciphertexts that are produced by the homomorphic evaluation algorithm, π is a succinct non-interactive argument that need not be zero-knowledge.

Specifically, the public key of the scheme consists of two public keys, pk_0 and pk_1 , of the underlying homomorphic scheme, a common reference string for a non-interactive zero-knowledge proof system, and t common reference strings for succinct non-interactive argument systems (where t is a predetermined upper bound on the number of repeated homomorphic operations that can be applied to a ciphertext produced by the encryption algorithm). The secret key consists of the corresponding secret keys sk_0 and sk_1 . For encrypting a message we encrypt it under each of pk_0 and pk_1 , and provide a non-interactive zero-knowledge proof that the resulting two ciphertexts are indeed encryptions of the same message. Thus, a ciphertext that is produced by the encryption algorithm has the form $(c_0, c_1, \pi_{\text{ZK}})$.

The homomorphic evaluation algorithm preserves the “double encryption” invariant. Specifically, given a ciphertext $(c_0, c_1, \pi_{\text{ZK}})$ that was produced by the encryption algorithm and a function $f \in \mathcal{F}$, the homomorphic evaluation algorithm first applies the homomorphic evaluation algorithm of the underlying encryption scheme to each of c_0 and c_1 . That is, it computes $c_0^{(1)} = \text{HomEval}_{pk_0}(c_0, f)$ and $c_1^{(1)} = \text{HomEval}_{pk_1}(c_1, f)$. Then, it computes a succinct non-interactive argument $\pi^{(1)}$ to the fact that there exist a function $f \in \mathcal{F}$ and a ciphertext $(c_0, c_1, \pi_{\text{ZK}})$, such that π_{ZK} is accepted by the verifier of the non-interactive zero-knowledge proof system, and that $c_0^{(1)}$ and $c_1^{(1)}$ are generated from c_0 and c_1 using f as specified. We denote the language of the corresponding argument system by $L^{(1)}$, and the resulting ciphertext is of the form $c^{(1)} = (1, c_0^{(1)}, c_1^{(1)}, \pi^{(1)})$. We point out that the usage of *succinct* arguments enables us to prevent the length of ciphertexts from increasing significantly.

More generally, given a ciphertext of the form $c^{(i)} = (i, c_0^{(i)}, c_1^{(i)}, \pi^{(i)})$, the homomorphic evaluation algorithm follows the same methodology for producing a ciphertext of the same form $c^{(i+1)} = (i+1, c_0^{(i+1)}, c_1^{(i+1)}, \pi^{(i+1)})$ using a succinct non-interactive argument system for a language $L^{(i+1)}$ stating that there exist a function $f \in \mathcal{F}$ and a ciphertext $c^{(i)}$ that is well-formed with respect to $L^{(i)}$, which were used for generating the current ciphertext $c^{(i+1)}$.

On the proof of security. Given an adversary that breaks the targeted malleability of our construction, we construct an adversary that breaks the security of (at least one of) the underlying building blocks. As in [NY90, DDN00, Sah99, Lin06], we show that this boils down to having a simulator that is able to decrypt a ciphertext while having access to only one of the secret keys sk_0 and sk_1 . This, in turn, enables the simulator to attack the public key pk_b for which sk_b is not known, where $b \in \{0, 1\}$. For satisfying our notion of security, however, such a simulator will not only have to decrypt a ciphertext, but to also recover a “certification chain” demonstrating that the ciphertext was produced by repeatedly applying the homomorphic evaluation algorithm. That is, given a well-formed ciphertext $c^{(i)} = (i, c_0^{(i)}, c_1^{(i)}, \pi^{(i)})$, the simulator needs to generate a “certification chain” for $c^{(i)}$ of the form $(c^{(0)}, f^{(0)}, \dots, c^{(i-1)}, f^{(i-1)}, c^{(i)})$, where:

1. $c^{(0)}$ is an output of the encryption algorithm, which can be decrypted while knowing only one of sk_0 and sk_1 .
2. For every $j \in \{1, \dots, i\}$ it holds that $c^{(j)}$ is obtained by applying the homomorphic evaluation

algorithm on $c^{(j-1)}$ and $f^{(j-1)}$.

For this purpose, we require that the argument systems used in the construction exhibit the following “knowledge extraction” property: for every efficient malicious prover P^* there exists an efficient “knowledge extractor” Ext_{P^*} , such that whenever P^* outputs a statement x and an argument π that are accepted by the verifier, Ext_{P^*} when given the random coins of P^* can in fact produce a witness w to the validity of x with all but a negligible probability.

By repeatedly applying such extractors the simulator is able to produce a certification chain. Then, given that the initial ciphertext $c^{(0)}$ is well-formed (i.e., the same messages is encrypted under pk_0 and pk_1), it can be decrypted using only one of the corresponding secret keys.

An alternative trade-off. In our first construction, the length of the ciphertext is essentially independent of t , and the public key consists of $t + 1$ common reference strings. In our second construction the number of common reference strings in the public key is only $\log t$, and a ciphertext now consists of $\log t$ ciphertexts of the underlying homomorphic scheme and $\log t$ succinct arguments. Such a trade-off may be preferable over the one offered by our first construction, for example, when using argument systems that are tailored to the \mathcal{NP} languages under considerations, or when it is not possible to use the same common reference string for all argument systems (depending, of course, on the length of the longest common reference strings).

The main idea underlying this construction is that the arguments computed by the homomorphic evaluation algorithm form a tree structure instead of a path structure. Specifically, instead of using t argument systems, we use only $d = \log t$ argument systems where the i -th one is used for arguing the well-formedness of a ciphertext after 2^i repeated homomorphic operations.

Succinct extractable arguments. As explained above, our construction hinges on the existence of succinct non-interactive argument systems that exhibit a knowledge extractor capable of extracting a witness from any successful prover. Gentry and Wichs [GW11] recently showed that no sub-linear non-interactive argument system can be proven secure by a black-box reduction to a falsifiable assumption. Fortunately, while we need succinct arguments, their lengths need not be sub-linear. It suffices for our purposes that arguments are shorter by only a multiplicative constant factor (say, $1/4$) than the length of the witness, and therefore the negative result of Gentry and Wichs does not apply to our settings. Nevertheless, all known argument systems that satisfy our needs are either set in the random oracle model or are based on non-falsifiable assumptions in the sense of Naor [Nao03].

The first such system was constructed by Micali [Mic00] using the PCP theorem. Computational soundness is proved in the random oracle model [BR93] and the length of the proofs is essentially independent of the length of the witness. Valiant [Val08] observed that the system is extractable as needed for our proof of security. Unfortunately, we inherently cannot use an argument system set in the random oracle model. To see why, consider a fresh ciphertext c which is an encryption of message m . After the first homomorphic operation we obtain a new ciphertext c' containing a proof π showing that c' is an encryption of $f(m)$ for some allowable function $f \in \mathcal{F}$. Verifying π requires access to the random oracle. Now, consider the second homomorphic operation resulting in c'' . The proof embedded in c'' must now prove, among other things, that there exists a valid proof π showing that c' is a well-formed ciphertext. But since π 's verifier queries the random oracle, this statement is in $\mathcal{NP}^{\mathcal{O}}$ where \mathcal{O} is a random oracle. Since PCPs do not relativize, it seems that Micali's system cannot be used for our purpose. In fact, there are no known succinct argument systems for proving statements in $\mathcal{NP}^{\mathcal{O}}$. This issue was also pointed out by Chiesa

and Tromer [CT10] in a completely different context, who suggested to overcome this difficulty by providing each prover with a smartcard implementing a specific oracle functionality.

Instead, we use a recent succinct non-interactive argument system due to Groth [Gro10] (see also the refinement by Lipmaa [Lip11]). Soundness is based on a variant of the “knowledge of exponent assumption,” a somewhat non-standard assumption (essentially stating that the required extractor exists by assumption, backed up with evidence in the generic group model). This class of assumptions was introduced by Damgård [Dam91] and extended by Bellare and Palacio [BP04b]. Interestingly, Bellare and Palacio [BP04a] succeeded in falsifying one such assumption using the Decision Diffie-Hellman problem. We note that while Groth’s argument system is even zero-knowledge, we primarily use the soundness property of the system (see the discussion in Section 6 on exploiting its zero-knowledge property).

1.3 Related Work

The problem of providing certain non-malleability properties for homomorphic encryption schemes was studied by Prabhakaran and Rosulek [PR08]. As a positive result, they presented a variant of the Cramer-Shoup encryption scheme [CS98] that provably supports linear operations and no other operations. There are two main differences between our work and the work of Prabhakaran and Rosulek: (1) their framework only considers sets of allowable functions that are *closed under composition*, and (2) their framework does not prevent ciphertext expansion during repeated applications of the homomorphic operation, whereas this is a key goal for our work.

In our work we do not make the assumption that the set of allowable functions \mathcal{F} is closed under composition. As already discussed, one of the advantages of avoiding this assumption (other than the obvious advantage of capturing a wider class of homomorphic schemes) is that we are in fact able to target the malleability of a scheme at any subset $\mathcal{F}' \subseteq \mathcal{F}$ of its supported homomorphic operations (which may be determined by the specific application in which the scheme is used), and this is especially significant when dealing with fully homomorphic schemes. Another advantage is the ability to limit the number of repeated homomorphic operations.

We note that when assuming that the set of functions \mathcal{F} is closed under composition, there is in fact a trivial solution: For encrypting a message m compute $(\text{Enc}_{pk}(m), \text{id})$ using any non-malleable encryption scheme, where id is the identity function. Then, the homomorphic evaluation algorithm on input a ciphertext (c, f_1) and a function $f_2 \in \mathcal{F}$ simply outputs $(c, f_2 \circ f_1)$ (where \circ denotes composition of functions). In this light, Prabhakaran and Rosulek focused on formalizing a meaningful notion of security for a-posteriori chosen-ciphertext attacks (CCA2), following previous relaxations of such attacks [ADR02, CKN03, Gro04, PR07]. This is orthogonal to our setting in which the issue of avoiding a blow-up in the length of the ciphertext makes the problem challenging already for chosen-plaintext attacks.

Finally, we note that targeted malleability shares a somewhat similar theme with the problem of outsourcing a computation in a verifiable manner from a computationally-weak client to a powerful server (see, for example, [GKR08, GGP10, CKV10, AIK10] and the references therein). In both settings the main goal from the security aspect is to guarantee that a “correct” or an “allowable” computation is performed. From the efficiency aspect, however, the two settings significantly differ: whereas for targeted malleability our main focus is to prevent a blow-up in the length of the ciphertext resulting from *repeated applications* of a computation, for verifiable computation the main focus is to minimize the client’s computational effort within a *single* computation.

1.4 Paper Organization

The remainder of this paper is organized as follows. In Section 2 we present the basic tools that are used in our constructions. In Section 3 we formalize the notion of targeted malleability. In Sections 4 and 5 we present our constructions. Finally, in Section 6 we discuss possible extensions of our work and several open problems.

2 Preliminaries

In this section we present the basic tools that are used in our constructions: public-key encryption and homomorphic encryption, succinct non-interactive arguments, and non-interactive zero-knowledge proofs.

2.1 Public-Key Encryption

A public-key encryption scheme is a triplet $\Pi = (\text{KeyGen}, \text{Enc}, \text{Dec})$ of probabilistic polynomial-time algorithms, where **KeyGen** is the key-generation algorithm, **Enc** is the encryption algorithm, and **Dec** is the decryption algorithm. The key-generation algorithm **KeyGen** receives as input the security parameter, and outputs a public key pk and a secret key sk . The encryption algorithm **Enc** receives as input a public key pk and a message m , and outputs a ciphertext c . The decryption algorithm **Dec** receives as input a ciphertext c and a secret key sk , and outputs a message m or the symbol \perp .

Functionality. In terms of functionality, in this paper we require the property of *almost-all-keys perfect decryption* [DNR04], defined as follows:

Definition 2.1. *A public-key encryption scheme $\Pi = (\text{KeyGen}, \text{Enc}, \text{Dec})$ has almost-all-keys perfect decryption if there exists a negligible function $\nu(k)$ such that for all sufficiently large k with probability $1 - \nu(k)$ over the choice of $(sk, pk) \leftarrow \text{KeyGen}(1^k)$, for any message m it holds that*

$$\Pr[\text{Dec}_{sk}(\text{Enc}_{pk}(m)) = m] = 1 \quad ,$$

where the probability is taken over the internal randomness of **Enc** and **Dec**.

We note that Dwork, Naor, and Reingold [DNR04] proposed a general transformation turning any encryption scheme into one that has almost-all-keys perfect decryption. When starting with a scheme that has a very low error probability, their transformation only changes the random bits used by the encryption algorithm, and in our setting this is important as it preserves the homomorphic operations. When starting with a scheme that has a significant error probability, we note that the error probability can be reduced exponentially by encrypting messages under several independently chosen public keys, and decrypting according to the majority. This again preserves the homomorphic operations.

Security. In terms of security, we consider the most basic notion of semantic-security against chosen-plaintext attacks, asking that any efficient adversary has only a negligible advantage in distinguishing between encryptions of different messages. This is formalized as follows:

Definition 2.2. *A public-key encryption scheme $\Pi = (\text{KeyGen}, \text{Enc}, \text{Dec})$ is semantically secure against chosen-plaintext attacks if for any probabilistic polynomial-time adversary $A = (A_1, A_2)$ it holds that*

$$\text{Adv}_{\Pi, A}^{\text{CPA}}(k) \stackrel{\text{def}}{=} \left| \Pr \left[\text{Expt}_{\Pi, A, 0}^{\text{CPA}}(k) = 1 \right] - \Pr \left[\text{Expt}_{\Pi, A, 1}^{\text{CPA}}(k) = 1 \right] \right|$$

is negligible in k , where $\text{Expt}_{\Pi, A, b}^{\text{CPA}}(k)$ is defined as follows:

1. $(sk, pk) \leftarrow \text{KeyGen}(1^k)$.
2. $(m_0, m_1, \text{state}) \leftarrow A_1(1^k, pk)$ such that $|m_0| = |m_1|$.
3. $c^* \leftarrow \text{Enc}_{pk}(m_b)$.
4. $b' \leftarrow A_2(c^*, \text{state})$
5. Output b' .

Homomorphic encryption. A public-key encryption scheme $\Pi = (\text{KeyGen}, \text{Enc}, \text{Dec})$ is *homomorphic* with respect to a set of efficiently computable functions \mathcal{F} if there exists a *homomorphic evaluation* algorithm HomEval that receives as input a public key pk , an encryption of a message m , and a function $f \in \mathcal{F}$, and outputs an encryption of the message $f(m)$. Formally, with overwhelming probability over the choice of $(sk, pk) \leftarrow \text{KeyGen}(1^k)$ (as in Definition 2.1), for any ciphertext c such that $\text{Dec}_{sk}(c) \neq \perp$ and for any function $f \in \mathcal{F}$ it holds that $\text{Dec}_{sk}(\text{HomEval}_{pk}(c, f)) = f(\text{Dec}_{sk}(c))$ with probability 1 over the internal randomness of HomEval and Dec .

The main property that is typically required from a homomorphic encryption scheme is *compactness*, asking that the length of the ciphertext does not trivially grow with the number of repeated homomorphic operations. In our setting, given an upper bound t on the number of repeated homomorphic operations that can be applied to a ciphertext produced by the encryption algorithm, we are interested in minimizing the dependency of the length of the ciphertext on t .

An additional property, that we do not consider in this paper, is of *function privacy*. Informally, this property asks that the homomorphic evaluation algorithm does not reveal which function from the set \mathcal{F} it receives as input. We refer the reader to [GHV10] for a formal definition. We note that in our setting, where function privacy is not taken into account, we can assume without loss of generality that the homomorphic evaluation algorithm is deterministic.

2.2 Non-Interactive Extractable Arguments

A non-interactive argument system for a language $L = \bigcup_{k \in \mathbb{N}} L(k)$ with a witness relation $R = \bigcup_{k \in \mathbb{N}} R(k)$ consists of a triplet of algorithms $(\text{CRSGen}, \text{P}, \text{V})$, where CRSGen is an algorithm generating a common reference string crs , and P and V are the prover and verifier algorithms, respectively. The prover takes as input a triplet (x, w, crs) , where $(x, w) \in R$, and outputs an argument π . The verifier takes as input a triplet (x, π, crs) and either accepts or rejects. In this paper we consider a setting where all three algorithms run in polynomial time, CRSGen and P may be probabilistic, and V is deterministic.

We require three properties from such a system. The first property is *perfect completeness*: for every (x, w, crs) such that $(x, w) \in R$, the prover always generates an argument that is accepted by the verifier. The second property is *knowledge extraction*: for every efficient malicious prover P^* there exists an efficient “knowledge extractor” Ext_{P^*} , such that whenever P^* outputs (x, π) that is accepted by the verifier, Ext_{P^*} when given the random coins of P^* can in fact produce a witness w such that $(x, w) \in R$ with all but a negligible probability. We note that this implies, in particular, soundness against efficient provers.

The perfect completeness and knowledge extraction properties are in fact trivial to satisfy: the prover can output the witness w as an argument, and the verifier checks that $(x, w) \in R$ (unlike for CS proofs [Mic00, Val08] we do not impose any non-trivial efficiency requirement on the verifier). The third property that we require from the argument system is that of having rather succinct

arguments: there should exist a constant $0 < \gamma < 1$ such that the arguments are of length at most $\gamma|w|$.

Definition 2.3. Let $0 < \gamma < 1$ be a constant. A γ -succinct non-interactive extractable argument system for a language $L = \bigcup_{k \in \mathbb{N}} L(k)$ with a witness relation $R_L = \bigcup_{k \in \mathbb{N}} R_{L(k)}$ is a triplet of probabilistic polynomial-time algorithms $(\text{CRSGen}, \text{P}, \text{V})$ with the following properties:

1. **Perfect completeness:** For every $k \in \mathbb{N}$ and $(x, w) \in R_{L(k)}$ it holds that

$$\Pr \left[\text{V}(1^k, x, \pi, \text{crs}) = 1 \mid \begin{array}{l} \text{crs} \leftarrow \text{CRSGen}(1^k) \\ \pi \leftarrow \text{P}(1^k, x, w, \text{crs}) \end{array} \right] = 1$$

where the probability is taken over the internal randomness of CRSGen , P and V .

2. **Adaptive knowledge extraction:** For every probabilistic polynomial-time algorithm P^* there exist a probabilistic polynomial-time algorithm Ext_{P^*} and a negligible function $\nu(\cdot)$ such that

$$\Pr \left[(x, w) \notin R_{L(k)} \text{ and } \text{V}(1^k, x, \pi, \text{crs}) = 1 \mid \begin{array}{l} \text{crs} \leftarrow \text{CRSGen}(1^k), r \leftarrow \{0, 1\}^* \\ (x, \pi) \leftarrow \text{P}^*(1^k, \text{crs}; r) \\ w \leftarrow \text{Ext}_{\text{P}^*}(1^k, \text{crs}, r) \end{array} \right] \leq \nu(k)$$

for all sufficiently large k , where the probability is taken over the internal randomness of CRSGen , P^* , V , and Ext_{P^*} .

3. **γ -Succinct arguments:** For every $k \in \mathbb{N}$, $(x, w) \in R_{L(k)}$ and $\text{crs} \in \{0, 1\}^*$, it holds that $\text{P}(1^k, x, w, \text{crs})$ produces a distribution over strings of length at most $\gamma|w|$.

Instantiation. An argument system satisfying Definition 2.3 (with a deterministic verifier) was recently constructed by Groth [Gro10] in the common-reference string model based on a certain “knowledge of exponent” assumption. His scheme is even zero-knowledge, and the length of the resulting arguments is essentially independent of the length of the witness. The length of the common-reference string, however, is at least quadratic in the length of the witness⁴, and this will limit our constructions to support only a constant number of repeated homomorphic operations. Any argument system satisfying Definition 2.3 with a common-reference string of length linear in the length of the witness will allow our first construction to support any logarithmic number of repeated homomorphic operations, and our second construction to support any polynomial number of such operations.

The running time of the knowledge extractor. The proofs of security of our constructions involve nested invocations of the knowledge extractors that are provided by Definition 2.3. When supporting only a constant number of repeated homomorphic operations the simulation will always run in polynomial time. When supporting a super-constant number of repeated homomorphic operations, we need to require that the knowledge extractor Ext_{P^*} corresponding to a malicious prover P^* runs in time that is linear in the running time of P^* . This (together with a common-reference string of linear length) will allow our first construction to support any logarithmic number of repeated homomorphic operations, and our second construction to support any polynomial number of such operations.

⁴For proving the satisfiability of a circuit of size s , the common-reference string in [Gro10] consists of $O(s^2)$ group elements, taken from a group where (in particular) the discrete logarithm problem is assumed to be hard. Lipmaa [Lip11] was able to slightly reduce the number of group elements, but even in his construction it is still super-linear.

2.3 Non-Interactive Simulation-Sound Adaptive Zero-Knowledge Proofs

We define the notion of a non-interactive simulation-sound adaptive zero-knowledge proof system [BFM88, FLS90, BSM⁺91, Sah99].

Definition 2.4. A non-interactive simulation-sound adaptive zero-knowledge proof system for a language $L = \bigcup_{k \in \mathbb{N}} L(k)$ with a witness relation $R_L = \bigcup_{k \in \mathbb{N}} R_{L(k)}$ is a tuple of probabilistic polynomial-time algorithms $\Pi = (\text{CRSGen}, P, V, S_1, S_2)$ with the following properties:

1. **Perfect completeness:** For every $k \in \mathbb{N}$ and $(x, w) \in R_{L(k)}$ it holds that

$$\Pr \left[V(1^k, x, \pi, \text{crs}) = 1 \mid \begin{array}{l} \text{crs} \leftarrow \text{CRSGen}(1^k) \\ \pi \leftarrow P(1^k, x, w, \text{crs}) \end{array} \right] = 1$$

where the probability is taken over the internal randomness of CRSGen , P and V .

2. **Adaptive soundness:** For every algorithm P^* there exists a negligible function $\nu(\cdot)$ such that

$$\Pr \left[x \notin L(k) \text{ and } V(1^k, x, \pi, \text{crs}) = 1 \mid \begin{array}{l} \text{crs} \leftarrow \text{CRSGen}(1^k) \\ (x, \pi) \leftarrow P^*(1^k, \text{crs}) \end{array} \right] \leq \nu(k)$$

for all sufficiently large k , where the probability is taken over the internal randomness of CRSGen , P^* , and V .

3. **Adaptive zero knowledge:** For every probabilistic polynomial-time algorithm A there exists a negligible function $\nu(\cdot)$ such that

$$\text{Adv}_{\Pi, A}^{\text{ZK}}(k) \stackrel{\text{def}}{=} \left| \Pr \left[\text{Expt}_{\Pi, A}^{\text{ZK}}(k) = 1 \right] - \Pr \left[\text{Expt}_{\Pi, A, S_1, S_2}^{\text{ZK}}(k) = 1 \right] \right| \leq \nu(k)$$

for all sufficiently large k , where the experiment $\text{Expt}_{\Pi, A}^{\text{ZK}}(k)$ is defined as:

- (a) $\text{crs} \leftarrow \text{CRSGen}(1^k)$
- (b) $b \leftarrow A^{P(1^k, \cdot, \cdot, \text{crs})}(1^k, \text{crs})$
- (c) Output b

and the experiment $\text{Expt}_{\Pi, A, S_1, S_2}^{\text{ZK}}(k)$ is defined as:

- (a) $(\text{crs}, \tau) \leftarrow S_1(1^k)$
- (b) $b \leftarrow A^{S'_2(1^k, \cdot, \cdot, \tau)}(1^k, \text{crs})$, where $S'_2(1^k, x, w, \tau) = S_2(1^k, x, \tau)$
- (c) output b

4. **Simulation soundness:** For every probabilistic polynomial-time algorithm A there exists a negligible function $\nu(\cdot)$ such that

$$\text{Adv}_{\Pi, A}^{\text{SS}}(k) \stackrel{\text{def}}{=} \Pr \left[\text{Expt}_{\Pi, A}^{\text{SS}}(k) = 1 \right] \leq \nu(k)$$

for all sufficiently large k , where the experiment $\text{Expt}_{\Pi, A}^{\text{SS}}(k)$ is defined as:

- (a) $(\text{crs}, \tau) \leftarrow S_1(1^k)$
- (b) $(x, \pi) \leftarrow A^{S_2(1^k, \cdot, \cdot, \tau)}(1^k, \text{crs})$
- (c) Denote by Q the set of S_2 's answers to A 's oracle queries
- (d) Output 1 if and only if $x \notin L(k)$, $\pi \notin Q$, and $V(1^k, x, \pi, \text{crs}) = 1$

3 Defining Targeted Malleability

In this section we introduce a framework for targeted malleability by formalizing *non-malleability with respect to a specific set of functions*. We begin by discussing the case of *univariate* functions, and then show that our approach naturally generalizes to the case of *multivariate* functions. Given an encryption scheme that is homomorphic with respect to a set of functions \mathcal{F} we would like to capture the following notion of security: For any efficient adversary that is given an encryption c of a message m and outputs an encryption c' of a message m' , it should hold that either (1) m' is independent of m , (2) $c' = c$ (and thus $m' = m$), or (3) c' is obtained by repeatedly applying the homomorphic evaluation algorithm on c using functions $f_1, \dots, f_\ell \in \mathcal{F}$. The first two properties are the standard ones for non-malleability [DDN00], and the third property captures targeted malleability.

Following [DDN00, BS99, PSV07] we formalize a simulation-based notion of security that compares a real-world adversary to a simulator that is not given any ciphertexts as input. Specifically, we consider two experiments: a real-world experiment, and a simulated experiment, and require that for any efficient real-world adversary there exists an efficient simulator such that the outputs of the two experiments are computationally indistinguishable⁵. We consider both chosen-plaintext attacks (CPA) and a-priori chosen-ciphertext attacks (CCA1). We assume that the set of functions \mathcal{F} is recognizable in polynomial time, and it may or may not be closed under composition.

Chosen-plaintext attacks (CPA). In the real-world experiment we consider adversaries that are described by two algorithms $A = (A_1, A_2)$. The algorithm A_1 takes as input the public key of the scheme, and outputs a description of a distribution \mathcal{M} over messages, a state information `state`₁ to be included in the output of the experiment, and a state information `state`₂ to be given as input to the algorithm A_2 . We note that `state`₁ and `state`₂ may include pk and \mathcal{M} . Then, the algorithm A_2 takes as input the state information `state`₂ and a sequence of ciphertexts that are encryptions of messages m_1, \dots, m_r sampled from \mathcal{M} . The algorithm A_2 outputs a sequence of ciphertexts c_1, \dots, c_q , and the output of the experiment is defined as $(\text{state}_1, m_1, \dots, m_r, d_1, \dots, d_q)$, where for every $j \in \{1, \dots, q\}$ the value d_j is one of two things: if c_j is equal to the i -th input ciphertext for some i then d_j is a special symbol `copy` _{i} ; otherwise d_j is the decryption of c_j .

In the simulated experiment the simulator is also described by two algorithms $S = (S_1, S_2)$. The algorithm S_1 takes as input the public key, and outputs a description of a distribution \mathcal{M} over messages, a state information `state`₁ to be included in the output of the experiment, and a state information `state`₂ to be given as input to the algorithm S_2 (as in the real world). Then, a sequence of messages is sampled from \mathcal{M} , but here the algorithm S_2 does not receive the encryptions of these messages, but only `state`₂. The algorithm S_2 should output q values, where each value can take one of three possible types. The first type is the special symbol `copy` _{i} , and in this case we define $d_j = \text{copy}_i$. This captures the ability of real-world adversary to copy one of the ciphertexts. The second type is an index $i \in \{1, \dots, r\}$ and a sequence of functions $f_1, \dots, f_\ell \in \mathcal{F}$, where ℓ is at most some predetermined upper bound t on the number of repeated homomorphic operations. In this case we define $d_j = f(m_i)$ where $f = f_1 \circ \dots \circ f_\ell$. This captures the ability of the real-world adversary to choose one of its input ciphertexts and apply the homomorphic evaluation algorithm for at most t times. The third type is a ciphertext c_j , and in this case d_j is defined as its decryption. As the simulator does not receive any ciphertexts as input, this captures the ability of the adversary to produce a ciphertext that is independent of its input ciphertexts. The output of the experiment

⁵As commented by Pass et al. [PSV07], note that a distinguisher between the two experiments corresponds to using a relation for capturing non-malleability as in [DDN00, BS99].

is defined as $(\text{state}_1, m_1, \dots, m_r, d_1, \dots, d_q)$.

Definition 3.1. Let $t = t(k)$ be a polynomial. A public-key encryption scheme $\Pi = (\text{KeyGen}, \text{Enc}, \text{Dec}, \text{HomEval})$ is t -bounded non-malleable against chosen-plaintext attacks with respect to a set of functions \mathcal{F} if for any polynomials $r = r(k)$ and $q = q(k)$ and for any probabilistic polynomial-time algorithm $A = (A_1, A_2)$ there exists a probabilistic polynomial-time algorithm $S = (S_1, S_2)$ such that the distributions $\{\text{Real}_{\Pi, A, t, r, q}^{\text{CPA}}(k)\}_{k \in \mathbb{N}}$ and $\{\text{Sim}_{\Pi, S, t, r, q}^{\text{CPA}}(k)\}_{k \in \mathbb{N}}$ (see Figure 1) are computationally indistinguishable.

$\text{Real}_{\Pi, A, t, r, q}^{\text{CPA}}(k)$:	$\text{Sim}_{\Pi, S, t, r, q}^{\text{CPA}}(k)$:
1. $(sk, pk) \leftarrow \text{KeyGen}(1^k)$	1. $(sk, pk) \leftarrow \text{KeyGen}(1^k)$
2. $(\mathcal{M}, \text{state}_1, \text{state}_2) \leftarrow A_1(1^k, pk)$	2. $(\mathcal{M}, \text{state}_1, \text{state}_2) \leftarrow S_1(1^k, pk)$
3. $(m_1, \dots, m_r) \leftarrow \mathcal{M}$	3. $(m_1, \dots, m_r) \leftarrow \mathcal{M}$
4. $c_i^* \leftarrow \text{Enc}_{pk}(m_i)$ for every $i \in \{1, \dots, r\}$	4. $(c_1, \dots, c_q) \leftarrow S_2(1^k, \text{state}_2)$
5. $(c_1, \dots, c_q) \leftarrow A_2(1^k, c_1^*, \dots, c_r^*, \text{state}_2)$	5. For every $j \in \{1, \dots, q\}$ let
6. For every $j \in \{1, \dots, q\}$ let	$d_j = \begin{cases} \text{copy}_i & \text{if } c_j = \text{copy}_i \\ & \text{if } c_j = (i, f_1, \dots, f_\ell) \\ & \quad \text{where } i \in \{1, \dots, r\}, \\ & \quad \ell \leq t, f_1, \dots, f_\ell \in \mathcal{F}, \\ & \quad \text{and } f = f_1 \circ \dots \circ f_\ell \\ \text{Dec}_{sk}(c_j) & \text{otherwise} \end{cases}$
$d_j = \begin{cases} \text{copy}_i & \text{if } c_j = c_i^* \\ \text{Dec}_{sk}(c_j) & \text{otherwise} \end{cases}$	
7. Output $(\text{state}_1, m_1, \dots, m_r, d_1, \dots, d_q)$	6. Output $(\text{state}_1, m_1, \dots, m_r, d_1, \dots, d_q)$

Figure 1: The distributions $\text{Real}_{\Pi, A, t, r, q}^{\text{CPA}}(k)$ and $\text{Sim}_{\Pi, S, t, r, q}^{\text{CPA}}(k)$.

Dealing with multivariate functions. Our approach naturally generalizes to the case of multivariate functions as follows. Fix a set \mathcal{F} of functions that are defined on d -tuples of plaintexts for some integer d , and let A be an efficient adversary that is given a sequence of ciphertexts c_1^*, \dots, c_r^* and outputs a sequence of ciphertexts c_1, \dots, c_q , as in Definition 3.1. Intuitively, for each output ciphertext c_j it should hold that either (1) $\text{Dec}_{sk}(c_j)$ is independent of c_1^*, \dots, c_r^* , (2) $c_j = c_i^*$ for some $i \in \{1, \dots, r\}$, or (3) c_j is obtained by repeatedly applying the homomorphic evaluation algorithm using functions from the set \mathcal{F} and a sequence of ciphertexts where each ciphertext is either taken from c_1^*, \dots, c_r^* or is independent of c_1^*, \dots, c_r^* .

Formally, the distribution $\text{Real}_{\Pi, A, t, r, q}^{\text{CPA}}(k)$ is not modified, and the distribution $\text{Sim}_{\Pi, S, t, r, q}^{\text{CPA}}(k)$ is modified by only changing the output $c_j = (i, f_1, \dots, f_\ell)$ of S_2 to a d -ary tree of depth at most t : each internal node contains a description of a function from the set \mathcal{F} , and each leaf contains either an index $i \in \{1, \dots, r\}$ or a plaintext m . The corresponding value d_j is then computed by evaluating the tree bottom-up where each index i is replaced by the plaintext m_i that was sampled from \mathcal{M} .

Dealing with randomized functions. The above definitions assume that \mathcal{F} is a set of *deterministic* functions. More generally, one can also consider *randomized* functions. There are two natural

approaches for extending our framework to this setting. The first approach is to view each function $f \in \mathcal{F}$ and string $r \in \{0,1\}^*$ (of an appropriate length) as defining a function $f_r(m) = f(m; r)$, and to apply the above definitions to the set $\mathcal{F}' = \{f_r\}_{f \in \mathcal{F}, r \in \{0,1\}^*}$ of deterministic functions. The second approach is to modify the distribution $\text{Sim}_{\Pi, S, t, r, q}^{\text{CPA}}(k)$ as follows: instead of setting d_j to the value $f(m_i)$, we sample d_j from the distribution induced by the random variable $f(m_i)$. Each of these two approaches may be preferable depending on the context in which the encryption scheme is used, and for simplifying the presentation in this paper we assume that \mathcal{F} is a set of deterministic functions.

A-priori chosen-ciphertexts attacks (CCA1). Definition 3.1 generalizes to a-priori chosen-ciphertext attacks by providing the algorithm A_1 oracle access to the decryption oracle before choosing the distribution \mathcal{M} . At the same time, however, the simulator still needs to specify the distribution \mathcal{M} without having such access (this is also known as *non-assisted simulation*). Specifically, we define $\{\text{Real}_{\Pi, A, t, r, q}^{\text{CCA1}}(k)\}_{k \in \mathbb{N}}$ and $\{\text{Sim}_{\Pi, S, t, r, q}^{\text{CCA1}}(k)\}_{k \in \mathbb{N}}$ as follows: $\text{Real}_{\Pi, A, t, r, q}^{\text{CCA1}}(k)$ is obtained from $\text{Real}_{\Pi, A, t, r, q}^{\text{CPA}}(k)$ by providing A_1 with oracle access to $\text{Dec}_{sk}(\cdot)$, and $\text{Sim}_{\Pi, S, t, r, q}^{\text{CCA1}}(k)$ is identical to $\text{Sim}_{\Pi, S, t, r, q}^{\text{CPA}}(k)$.

Definition 3.2. Let $t = t(k)$ be a polynomial. A public-key encryption scheme $\Pi = (\text{KeyGen}, \text{Enc}, \text{Dec}, \text{HomEval})$ is t -bounded non-malleable against a-priori chosen-ciphertext attacks with respect to a set of functions \mathcal{F} if for any polynomials $r = r(k)$ and $q = q(k)$ and for any probabilistic polynomial-time algorithm $A = (A_1, A_2)$ there exists a probabilistic polynomial-time algorithm $S = (S_1, S_2)$ such that the distributions $\{\text{Real}_{\Pi, A, t, r, q}^{\text{CCA1}}(k)\}_{k \in \mathbb{N}}$ and $\{\text{Sim}_{\Pi, S, t, r, q}^{\text{CCA1}}(k)\}_{k \in \mathbb{N}}$ are computationally indistinguishable.

4 The Path-Based Construction

In this section we present our first construction. The construction is based on any public-key encryption scheme that is homomorphic with respect to some set \mathcal{F} of functions, a non-interactive zero-knowledge proof system, and γ -succinct non-interactive argument systems for $\gamma = 1/4$. The scheme is parameterized by an upper bound t on the number of repeated homomorphic operations that can be applied to a ciphertext produced by the encryption algorithm. The scheme enjoys the feature that the dependency on t is essentially eliminated from the length of the ciphertext, and shifted to the public key. The public key consists of $t + 1$ common reference strings: one for the zero-knowledge proof system, and t for the succinct argument systems. We note that in various cases (such as argument systems in the common *random* string model) it may be possible to use only one common-reference string for all t argument systems, and then the length of the public key decreases quite significantly.

In Section 4.1 we formally specify the building blocks of the scheme, and in Section 4.2 we provide a description of the scheme. In Section 4.3 we prove the security of the scheme against chosen-plaintexts attacks (CPA), and in Section 4.4 we show that the proof in fact extends to deal with a-priori chosen-ciphertext attacks (CCA1).

4.1 The Building Blocks

Our construction relies on the following building blocks:

1. A homomorphic public-key encryption scheme $\Pi = (\text{KeyGen}, \text{Enc}, \text{Dec}, \text{HomEval})$ with respect to an efficiently recognizable set of efficiently computable functions \mathcal{F} . We assume that the

scheme has almost-all-keys perfect decryption (see Definition 2.1). In addition, as discussed in Section 2.1, as we do not consider function privacy we assume without loss of generality that **HomEval** is deterministic.

For any security parameter $k \in \mathbb{N}$ we denote by $\ell_{pk} = \ell_{pk}(k)$, $\ell_m = \ell_m(k)$, $\ell_r = \ell_r(k)$, and $\ell_c = \ell_c(k)$ the bit-lengths of the public key, plaintext, randomness of **Enc**, and ciphertext⁶, respectively, for the scheme Π . In addition, we denote by $V_{\mathcal{F}}$ the deterministic polynomial-time algorithm for testing membership in the set \mathcal{F} , and denote by $\ell_{\mathcal{F}} = \ell_{\mathcal{F}}(k)$ the bit-length of the description of each function $f \in \mathcal{F}$.

2. A non-interactive deterministic-verifier simulation-sound adaptive zero-knowledge proof system (see Section 2.3) $\Pi^{(0)} = (\text{CRSGen}^{(0)}, \text{P}^{(0)}, \text{V}^{(0)})$ for the \mathcal{NP} -language $L^{(0)} = \bigcup_{k \in \mathbb{N}} L^{(0)}(k)$ defined as follows.

$$L^{(0)}(k) = \left\{ \begin{array}{l} \exists (m, r_0, r_1) \in \{0, 1\}^{\ell_m + 2\ell_r} \text{ s.t.} \\ (pk_0, pk_1, c_0^{(0)}, c_1^{(0)}) \in \{0, 1\}^{2\ell_{pk} + 2\ell_c} : \\ \quad c_0^{(0)} = \text{Enc}_{pk_0}(m; r_0) \\ \quad \text{and } c_1^{(0)} = \text{Enc}_{pk_1}(m; r_1) \end{array} \right\}$$

For any security parameter $k \in \mathbb{N}$ we denote by $\ell_{\text{crs}^{(0)}} = \ell_{\text{crs}^{(0)}}(k)$ and $\ell_{\pi^{(0)}} = \ell_{\pi^{(0)}}(k)$ the bit-lengths of the common-reference strings produced by $\text{CRSGen}^{(0)}$ and of the proofs produced by $\text{P}^{(0)}$, respectively. Without loss of generality we assume that $\ell_{\pi^{(0)}} \geq \max\{\ell_c, \ell_{\mathcal{F}}\}$ (as otherwise proofs can always be padded).

3. For every $i \in \{1, \dots, t\}$ a $1/4$ -succinct non-interactive deterministic-verifier extractable argument system (see Section 2.2) $\Pi^{(i)} = (\text{CRSGen}^{(i)}, \text{P}^{(i)}, \text{V}^{(i)})$ for the \mathcal{NP} -language $L^{(i)} = \bigcup_{k \in \mathbb{N}} L^{(i)}(k)$ defined as follows.

$$L^{(i)}(k) = \left\{ \begin{array}{l} (pk_0, pk_1, c_0^{(i)}, c_1^{(i)}, \text{crs}^{(i-1)}, \dots, \text{crs}^{(0)}) \in \{0, 1\}^{2\ell_{pk} + 2\ell_c + \sum_{j=0}^{i-1} \ell_{\text{crs}^{(j)}}} : \\ \exists (c_0^{(i-1)}, c_1^{(i-1)}, \pi^{(i-1)}, f) \in \{0, 1\}^{2\ell_c + \ell_{\pi^{(i-1)}} + \ell_{\mathcal{F}}} \text{ s.t.} \\ \quad \bullet \ V_{\mathcal{F}}(f) = 1 \\ \quad \bullet \ c_0^{(i)} = \text{HomEval}_{pk_0}(c_0^{(i-1)}, f) \\ \quad \bullet \ c_1^{(i)} = \text{HomEval}_{pk_1}(c_1^{(i-1)}, f) \\ \quad \bullet \ \text{V}^{(i-1)}((pk_0, pk_1, c_0^{(i-1)}, c_1^{(i-1)}, \text{crs}^{(i-2)}, \dots, \text{crs}^{(0)}), \pi^{(i-1)}, \text{crs}^{(i-1)}) = 1 \end{array} \right\}$$

For any security parameter $k \in \mathbb{N}$ we denote by $\ell_{\text{crs}^{(i)}} = \ell_{\text{crs}^{(i)}}(k)$ and $\ell_{\pi^{(i)}} = \ell_{\pi^{(i)}}(k)$ the bit-lengths of the common-reference strings produced by $\text{CRSGen}^{(i)}$ and of the arguments produced by $\text{P}^{(i)}$, respectively.

4.2 The Scheme

The scheme $\Pi' = (\text{KeyGen}', \text{Enc}', \text{Dec}', \text{HomEval}')$ is parameterized by an upper bound t on the number of repeated homomorphic operations that can be applied to a ciphertext produced by the encryption algorithm. The scheme is defined as follows:

⁶For simplicity we assume a fixed upper bound ℓ_c on the length of ciphertexts, but this is not essential to our construction. More generally, one can allow the length of ciphertexts to increase as a result of applying the homomorphic operation.

- **Key generation:** On input 1^k sample two pairs of keys $(sk_0, pk_0) \leftarrow \text{KeyGen}(1^k)$ and $(sk_1, pk_1) \leftarrow \text{KeyGen}(1^k)$. Then, for every $i \in \{0, \dots, t\}$ sample $\text{crs}^{(i)} \leftarrow \text{CRSGen}^{(i)}(1^k)$. Output the secret key $sk = (sk_0, sk_1)$ and the public key $pk = (pk_0, pk_1, \text{crs}^{(0)}, \dots, \text{crs}^{(t)})$.
- **Encryption:** On input a public key pk and a plaintext m , sample $r_0, r_1 \in \{0, 1\}^*$ uniformly at random, and output the ciphertext $c^{(0)} = (0, c_0^{(0)}, c_1^{(0)}, \pi^{(0)})$, where

$$\begin{aligned} c_0^{(0)} &= \text{Enc}_{pk_0}(m; r_0) , \\ c_1^{(0)} &= \text{Enc}_{pk_1}(m; r_1) , \\ \pi^{(0)} &\leftarrow \text{P}^{(0)} \left((pk_0, pk_1, c_0^{(0)}, c_1^{(0)}), (m, r_0, r_1), \text{crs}^{(0)} \right) . \end{aligned}$$

- **Homomorphic evaluation:** On input a public key pk , a ciphertext $(i, c_0^{(i)}, c_1^{(i)}, \pi^{(i)})$, and a function $f \in \mathcal{F}$, proceed as follows. If $i \notin \{0, \dots, t-1\}$ or

$$\text{V}^{(i)} \left((pk_0, pk_1, c_0^{(i)}, c_1^{(i)}, \text{crs}^{(i-1)}, \dots, \text{crs}^{(0)}), \pi^{(i)}, \text{crs}^{(i)} \right) = 0$$

then output \perp . Otherwise, output the ciphertext $c^{(i+1)} = (i+1, c_0^{(i+1)}, c_1^{(i+1)}, \pi^{(i+1)})$, where

$$\begin{aligned} c_0^{(i+1)} &= \text{HomEval}_{pk_0}(c_0^{(i)}, f) , \\ c_1^{(i+1)} &= \text{HomEval}_{pk_1}(c_1^{(i)}, f) , \\ \pi^{(i+1)} &\leftarrow \text{P}^{(i+1)} \left((pk_0, pk_1, c_0^{(i+1)}, c_1^{(i+1)}, \text{crs}^{(i)}, \dots, \text{crs}^{(0)}), (c_0^{(i)}, c_1^{(i)}, \pi^{(i)}, f), \text{crs}^{(i+1)} \right) . \end{aligned}$$

- **Decryption:** On input a secret key sk and a ciphertext $(i, c_0^{(i)}, c_1^{(i)}, \pi^{(i)})$, output \perp if $i \notin \{0, \dots, t\}$ or

$$\text{V}^{(i)} \left((pk_0, pk_1, c_0^{(i)}, c_1^{(i)}, \text{crs}^{(i-1)}, \dots, \text{crs}^{(0)}), \pi^{(i)}, \text{crs}^{(i)} \right) = 0 .$$

Otherwise, compute $m_0 = \text{Dec}_{sk_0}(c_0^{(i)})$ and $m_1 = \text{Dec}_{sk_1}(c_1^{(i)})$. If $m_0 \neq m_1$ then output \perp , and otherwise output m_0 .

Note that at any point in time a ciphertext of the scheme is of the form $(i, c_0^{(i)}, c_1^{(i)}, \pi^{(i)})$, where $i \in \{0, \dots, t\}$, $c_0^{(i)}$ and $c_1^{(i)}$ are ciphertexts of the underlying encryption scheme, and $\pi^{(i)}$ is a proof or an argument with respect to one of $\Pi^{(0)}, \dots, \Pi^{(t)}$. Note that the assumption that the argument systems $\Pi^{(1)}, \dots, \Pi^{(t)}$ are 1/4-succinct implies that the length of their arguments is upper bounded by length of the proofs of $\Pi^{(0)}$ (i.e., $\ell_{\pi^{(i)}} \leq \ell_{\pi^{(0)}}$ for every $i \in \{1, \dots, t\}$). Thus, the only dependency on t in the length of the ciphertext results from the $\lceil \log_2(t+1) \rceil$ bits describing the prefix i .

4.3 Chosen-Plaintext Security

We now prove that the construction offers targeted malleability against chosen-plaintext attacks. For concreteness we focus on the case of a single message and a single ciphertext (i.e., the case $r(k) = q(k) = 1$ in Definition 3.1), and note that the more general case is a straightforward generalization. Given an adversary $A = (A_1, A_2)$ we construct a simulator $S = (S_1, S_2)$ as follows.

The algorithm S_1 . The algorithm S_1 is identical to A_1 , except for also including the public key pk and the distribution \mathcal{M} in the state that it forwards to S_2 . That is, S_1 on input $(1^k, pk)$ invokes A_1 on the same input to obtain a triplet $(\mathcal{M}, \text{state}_1, \text{state}_2)$, and then outputs $(\mathcal{M}, \text{state}_1, \text{state}'_2)$ where $\text{state}'_2 = (pk, \mathcal{M}, \text{state}_2)$.

The algorithm S_2 . The algorithm S_2 on input $(1^k, \text{state}'_2)$ where $\text{state}'_2 = (pk, \mathcal{M}, \text{state}_2)$, first samples $m' \leftarrow \mathcal{M}$, and computes $c^* \leftarrow \text{Enc}'_{pk}(m')$. Then, it samples $r \leftarrow \{0, 1\}^*$, and computes $c = (i, c_0^{(i)}, c_1^{(i)}, \pi^{(i)}) = A_2(1^k, c^*, \text{state}_2; r)$. If $i \notin \{0, \dots, t\}$ or

$$\mathbf{V}^{(i)} \left((pk_0, pk_1, c_0^{(i)}, c_1^{(i)}, \text{crs}^{(i-1)}, \dots, \text{crs}^{(0)}), \pi^{(i)}, \text{crs}^{(i)} \right) = 0$$

then S_2 outputs c . Otherwise, S_2 utilizes the knowledge extractors guaranteed by the argument systems $\Pi^{(1)}, \dots, \Pi^{(t)}$ to generate a “certification chain” for c of the form

$$(c^{(0)}, f^{(0)}, \dots, c^{(i-1)}, f^{(i-1)}, c^{(i)})$$

satisfying the following two properties:

1. $c^{(i)} = c$.
2. For every $j \in \{1, \dots, i\}$ it holds that $c^{(j)} = \text{HomEval}'_{pk}(c^{(j-1)}, f^{(j-1)})$.

We elaborate below on the process of generating the certification chain. If S_2 fails in generating such a chain then it outputs c . Otherwise, S_2 computes its output as follows:

1. If $c^{(0)} = c^*$ and $i = 0$, then S_2 outputs copy_1 .
2. If $c^{(0)} = c^*$ and $i > 0$, then S_2 outputs $f^{(0)} \circ \dots \circ f^{(i-1)}$.
3. If $c^{(0)} \neq c^*$, then S_2 outputs c .

Generating the certification chain. We say that a ciphertext $(i, c_0^{(i)}, c_1^{(i)}, \pi^{(i)})$ is *valid* if $i \in \{0, \dots, t\}$ and

$$\mathbf{V}^{(i)} \left((pk_0, pk_1, c_0^{(i)}, c_1^{(i)}, \text{crs}^{(i-1)}, \dots, \text{crs}^{(0)}), \pi^{(i)}, \text{crs}^{(i)} \right) = 1 \ .$$

Viewing the algorithm A_2 as a malicious prover with respect to the argument system $\Pi^{(i)}$ with the common reference string $\text{crs}^{(i)}$, whenever A_2 outputs a valid ciphertext $c^{(i)} = (i, c_0^{(i)}, c_1^{(i)}, \pi^{(i)})$, the algorithm S_2 invokes the knowledge extractor Ext_{A_2} that corresponds to A_2 (recall Definition 2.3) to obtain a witness $(c_0^{(i-1)}, c_1^{(i-1)}, \pi^{(i-1)}, f^{(i-1)})$ to the fact that

$$(pk_0, pk_1, c_0^{(i)}, c_1^{(i)}, \text{crs}^{(i-1)}, \dots, \text{crs}^{(0)}) \in L^{(i)} \ .$$

Note that S_2 chooses the randomness for A_2 , which it can then provide to Ext_{A_2} . If successful then by the definition of $L^{(i)}$ we have a new valid ciphertext $c^{(i-1)} = (i-1, c_0^{(i-1)}, c_1^{(i-1)}, \pi^{(i-1)})$. If $i = 1$ then we are done. Otherwise (i.e., if $i > 1$), viewing the combination of A_2 and Ext_{A_2} as a malicious prover with respect to the argument system $\Pi^{(i-1)}$ with the common reference string $\text{crs}^{(i-1)}$, the

algorithm S_2 invokes the knowledge extractor $\text{Ext}_{(A_2, \text{Ext}_{A_2})}$ that corresponds to the combination of A_2 and Ext_{A_2} to obtain a witness $(c_0^{(i-2)}, c_1^{(i-2)}, \pi^{(i-2)}, f^{(i-2)})$ to the fact that

$$(pk_0, pk_1, c_0^{(i-1)}, c_1^{(i-1)}, \text{crs}^{(i-2)}, \dots, \text{crs}^{(0)}) \in L^{(i-1)},$$

and so on for i iterations or until the first failure.

Having described the simulator we now prove the following theorem stating the security of the scheme in the case $r(k) = q(k) = 1$ (noting again that the more general case is a straightforward generalization). As discussed in Section 2.2, the quadratic blow-up in the length of the common-reference string in Groth's argument system [Gro10] restricts our treatment here to a constant number t of repeated homomorphic operations, and any improvement to Groth's argument system with a common-reference string of linear length will directly allow any logarithmic number of repeated homomorphic operations (and any polynomial number of such operations in the scheme presented in Section 5).

Theorem 4.1. *For any constant $t \in \mathbb{N}$ and for any probabilistic polynomial-time adversary A the distributions $\{\text{Real}_{\Pi', A, t, r, q}^{\text{CPA}}(k)\}_{k \in \mathbb{N}}$ and $\{\text{Sim}_{\Pi', S, t, r, q}^{\text{CPA}}(k)\}_{k \in \mathbb{N}}$ are computationally indistinguishable, for $r(k) = q(k) = 1$.*

Proof. We define a sequence of distributions $\mathcal{D}_1, \dots, \mathcal{D}_7$ such that $\mathcal{D}_1 = \text{Sim}_{\Pi', S, t, r, q}^{\text{CPA}}$ and $\mathcal{D}_7 = \text{Real}_{\Pi', A, t, r, q}^{\text{CPA}}$, and prove that for every $i \in \{1, \dots, 6\}$ the distributions \mathcal{D}_i and \mathcal{D}_{i+1} are computationally indistinguishable. For simplicity in what follows we assume that the scheme $\Pi = (\text{KeyGen}, \text{Enc}, \text{Dec}, \text{HomEval})$ actually has perfect decryption for all keys (and not with an overwhelming probability over the choice of keys). This assumption clearly does not hurt any of the indistinguishability arguments in our proof, since we can initially condition on the event that both (sk_0, pk_0) and (sk_1, pk_1) provide perfect decryption.

The distribution \mathcal{D}_1 . This is the distribution $\text{Sim}_{\Pi', S, t, r, q}^{\text{CPA}}$.

The distribution \mathcal{D}_2 . This distribution is obtained from \mathcal{D}_1 via the following modification. As in \mathcal{D}_1 , if S_2 fails to obtain a certification chain, then output $(\text{state}_1, m, \perp)$. Otherwise, the output is computed as follows:

1. If $c^{(0)} = c^*$ and $i = 0$ then output $(\text{state}_1, m, \text{copy}_1)$. This is identical to \mathcal{D}_1 .
2. If $c^{(0)} = c^*$ and $i > 0$ then output $(\text{state}_1, m, f(m))$, where $f = f^{(0)} \circ \dots \circ f^{(i-1)}$. This is identical to \mathcal{D}_1 .
3. If $c^{(0)} \neq c^*$ then compute the message $m^{(0)} = \text{Dec}'_{sk}(c^{(0)})$. If $m^{(0)} \neq \perp$ then output $(\text{state}_1, m, f(m^{(0)}))$, where $f = f^{(0)} \circ \dots \circ f^{(i-1)}$, and otherwise output $(\text{state}_1, m, \perp)$. That is, in this case instead of invoking the decryption algorithm Dec' on $c^{(i)}$, we invoke it on $c^{(0)}$, and then apply the functions given by the certification chain.

The distribution \mathcal{D}_3 . This distribution is obtained from \mathcal{D}_2 by producing $\text{crs}^{(0)}$ and π^* (where $c^* = (c_0^*, c_1^*, \pi^*)$) using the simulator of the NIZK proof system $\Pi^{(0)}$.

The distribution \mathcal{D}_4 . This distribution is obtained from \mathcal{D}_3 by producing the challenge ciphertext $c^* = (c_0^*, c_1^*, \pi^*)$ with $c_0^* = \text{Enc}_{pk_0}(m)$ (instead of $c_0^* = \text{Enc}_{pk_0}(m')$ as in \mathcal{D}_3).

The distribution \mathcal{D}_5 . This distribution is obtained from \mathcal{D}_4 by producing the challenge ciphertext $c^* = (c_0^*, c_1^*, \pi^*)$ with $c_1^* = \text{Enc}_{pk_1}(m)$ (instead of $c_1^* = \text{Enc}_{pk_1}(m')$ as in \mathcal{D}_4).

The distribution \mathcal{D}_6 . This distribution is obtained from \mathcal{D}_5 by producing $\text{crs}^{(0)}$ and π^* (where $c^* = (c_0^*, c_1^*, \pi^*)$) using the algorithms $\text{CRSGen}^{(0)}$ and $\text{P}^{(0)}$, respectively (and not by using the simulator of the NIZK proof system $\Pi^{(0)}$ as in \mathcal{D}_5).

The distribution \mathcal{D}_7 . This is the distribution $\text{Real}_{\Pi', A, t, r, q}^{\text{CPA}}$.

Before proving that the above distributions are computationally indistinguishable, we first prove that S_2 fails to produce a certification chain with all but a negligible probability.

Lemma 4.2. *In distributions $\mathcal{D}_1, \dots, \mathcal{D}_6$, whenever A_2 outputs a valid ciphertext, S_2 generates a certification chain with all but a negligible probability.*

Proof. Assume towards a contradiction that in one of $\mathcal{D}_1, \dots, \mathcal{D}_6$ with a non-negligible probability it holds that A_2 outputs a valid ciphertext but S_2 fails to generate a certification chain. In particular, there exists an index $i \in \{1, \dots, t\}$ for which with a non-negligible probability A_2 outputs a valid ciphertext of the form $c^{(i)} = (i, c_0^{(i)}, c_1^{(i)}, \pi^{(i)})$, but S_2 fails to generate a certification chain. Recall that for generating a certification chain starting with $c^{(i)}$, the simulator S_2 attempts to invoke i knowledge extractors (until the first failure occurs) that we denote by $\text{Ext}^{(i)}, \dots, \text{Ext}^{(1)}$. These knowledge extractors correspond to the malicious provers described in the description of S_2 for the argument systems $\Pi^{(i)}, \dots, \Pi^{(1)}$, respectively. Then, there exists an index $j \in \{1, \dots, i\}$ for which with a non-negligible probability S_2 is successful with $\text{Ext}^{(i)}, \dots, \text{Ext}^{(j+1)}$ but fails with $\text{Ext}^{(j)}$. The fact that S_2 is successful with $\text{Ext}^{(j+1)}$ implies that it produces a valid ciphertext $c^{(j)} = (j, c_0^{(j)}, c_1^{(j)}, \pi^{(j)})$. In particular, it holds that

$$\text{V}^{(j)} \left((pk_0, pk_1, c_0^{(j)}, c_1^{(j)}, \text{crs}^{(j-1)}, \dots, \text{crs}^{(0)}), \pi^{(j)}, \text{crs}^{(j)} \right) = 1.$$

Now, the fact that with a non-negligible probability S_2 fails with $\text{Ext}^{(j)}$ immediately translates to a malicious prover that contradicts the knowledge extraction property of the argument system $\Pi^{(j)}$. ■

We now prove that for every $i \in \{1, \dots, 6\}$ the distributions \mathcal{D}_i and \mathcal{D}_{i+1} are computationally indistinguishable.

Lemma 4.3. *The distributions \mathcal{D}_1 and \mathcal{D}_2 are computationally indistinguishable.*

Proof. Whenever A_2 outputs an invalid ciphertext, or outputs a valid ciphertext and S_2 generates a certification chain, the distributions \mathcal{D}_1 and \mathcal{D}_2 are identical. Indeed, in such a case the perfect decryption property guarantees that $\text{Dec}'_{sk}(c^{(i)}) = f(m^{(0)})$. Therefore, \mathcal{D}_1 and \mathcal{D}_2 differ only when A_2 outputs a valid ciphertext but S_2 fails to generate a certification chain. Lemma 4.2 guarantees that this event occurs with only a negligible probability. ■

Lemma 4.4. *The distributions \mathcal{D}_2 and \mathcal{D}_3 are computationally indistinguishable.*

Proof. This follows from the zero-knowledge property of $\Pi^{(0)}$. Specifically, any efficient algorithm that distinguishes between \mathcal{D}_2 and \mathcal{D}_3 can be used (together with S) in a straightforward manner to contradict the zero-knowledge property of $\Pi^{(0)}$. ■

Lemma 4.5. *The distributions \mathcal{D}_3 and \mathcal{D}_4 are computationally indistinguishable.*

Proof. The simulation soundness of $\Pi^{(0)}$ guarantees that instead of computing $\text{Dec}'_{sk}(c^{(0)})$ we can verify that $V^{(0)}\left(\left(pk_0, pk_1, c_0^{(0)}, c_1^{(0)}\right), \pi^{(0)}, \text{crs}^{(0)}\right) = 1$, and then compute $\text{Dec}_{sk_1}(c_1^{(0)})$. The resulting distribution will be identical with all but a negligible probability. This implies that we do not need the key sk_0 , and this immediately translates to a distinguisher between $(pk_0, \text{Enc}_{pk_0}(m))$ and $(pk_0, \text{Enc}_{pk_0}(m'))$, where m and m' are sampled independently from \mathcal{M} . That is, the simulation soundness of $\Pi^{(0)}$ and the semantic security of the underlying encryption scheme guarantee that \mathcal{D}_3 and \mathcal{D}_4 are computationally indistinguishable. ■

Lemma 4.6. *The distributions \mathcal{D}_4 and \mathcal{D}_5 are computationally indistinguishable.*

Proof. As in the proof of Lemma 4.5, the simulation soundness of $\Pi^{(0)}$ guarantees that instead of computing $\text{Dec}'_{sk}(c^{(0)})$ we can verify that $V^{(0)}\left(\left(pk_0, pk_1, c_0^{(0)}, c_1^{(0)}\right), \pi^{(0)}, \text{crs}^{(0)}\right) = 1$, and then compute $\text{Dec}_{sk_0}(c_0^{(0)})$. The resulting distribution will be identical with all but a negligible probability. This implies that we do not need the key sk_1 , and this immediately translates to a distinguisher between $(pk_1, \text{Enc}_{pk_1}(m))$ and $(pk_1, \text{Enc}_{pk_1}(m'))$, where m and m' are sampled independently from \mathcal{M} . That is, the simulation soundness of $\Pi^{(0)}$ and the semantic security of the underlying encryption scheme guarantee that \mathcal{D}_4 and \mathcal{D}_5 are computationally indistinguishable. ■

Lemma 4.7. *The distributions \mathcal{D}_5 and \mathcal{D}_6 are computationally indistinguishable.*

Proof. As in the proof of Lemma 4.4, this follows from the zero-knowledge property of $\Pi^{(0)}$. Specifically, any efficient algorithm that distinguishes between \mathcal{D}_5 and \mathcal{D}_6 can be used (together with S) in a straightforward manner to contradict the zero-knowledge property of $\Pi^{(0)}$. ■

Lemma 4.8. *The distributions \mathcal{D}_6 and \mathcal{D}_7 are computationally indistinguishable.*

Proof. In the distributions \mathcal{D}_6 and \mathcal{D}_7 the algorithm A_2 receives an encryption c^* of m , and outputs a ciphertext c . First, we note that in both \mathcal{D}_6 and \mathcal{D}_7 , if $c = c^*$ then the output is $(\text{state}_1, m, \text{copy}_1)$, and if c is invalid then the output is $(\text{state}_1, m, \perp)$. Therefore, we now focus on the case that $c \neq c^*$ and c is valid. In this case, in \mathcal{D}_7 the output is $(\text{state}_1, m, \text{Dec}'_{sk}(c))$, and we now show that with an overwhelming probability the same output is obtained also in \mathcal{D}_6 .

In \mathcal{D}_6 Lemma 4.2 guarantees that whenever c is valid S_2 produces a certification chain with all but a negligible probability. There are now two cases to consider. In the first case, if $c^{(0)} = c^*$ and $i > 0$ then the output of \mathcal{D}_6 is $(\text{state}_1, m, f(m))$, where $f = f^{(0)} \circ \dots \circ f^{(i-1)}$. Since c^* is in fact an encryption of m , then the perfect decryption property guarantees that $\text{Dec}'_{sk}(c) = f(m)$. In the second case, if $c^{(0)} \neq c^*$ then the output of \mathcal{D}_6 is $(\text{state}_1, m, f(m^{(0)}))$ where $m^{(0)} = \text{Dec}'_{sk}(c^{(0)})$. Again by the perfect decryption property, it holds that $\text{Dec}'_{sk}(c) = f(m^{(0)})$. ■

This concludes the proof of Theorem 4.1. ■

4.4 Chosen-Ciphertext Security

We now show that the proof of security in Section 4.3 in fact extends to the setting of a-priori chosen-ciphertext attacks (CCA1). The difficulty in extending the proof is that now whereas the adversary A_1 is given oracle access to the decryption algorithm, the simulator S_1 is not given such access. Therefore, it is not immediately clear that S_1 can correctly simulate the decryption queries of A_1 . We note that this issue seems to capture the main difference between the simulation-based and the indistinguishability-based approaches for defining non-malleability, as pointed out by Bellare and Sahai [BS99].

We resolve this issue using the approach of [DDN00, BS99]: S will not run A on the given public key pk , but instead will sample a new public key pk' together with a corresponding secret key sk' , and run A on pk' . This way, S can use the secret key sk' for answering all of A_1 's decryption queries. In addition, when A_2 outputs a ciphertext, S then uses sk' for “translating” this ciphertext from pk' to pk .

We now provide the modified description of S . Given an adversary $A = (A_1, A_2)$ we define the simulator $S = (S_1, S_2)$ as follows.

The algorithm S_1 . The algorithm S_1 on input $(1^k, pk)$ first samples $(sk', pk') \leftarrow \text{KeyGen}'(1^k)$. Then, it invokes A_1 on the input $(1^k, pk')$ while answering decryption queries using the secret key sk' , to obtain a triplet $(\mathcal{M}, \text{state}_1, \text{state}_2)$. Finally, it outputs $(\mathcal{M}, \text{state}_1, \text{state}_2')$ where $\text{state}_2' = (\mathcal{M}, pk, sk', pk', \text{state}_2)$.

The algorithm S_2 . The algorithm S_2 on input $(1^k, \text{state}_2')$ where $\text{state}_2' = (\mathcal{M}, pk, sk', pk', \text{state}_2)$, first samples $m' \leftarrow \mathcal{M}$ and computes $c^* \leftarrow \text{Enc}'_{pk'}(m')$. Then, it samples $r \leftarrow \{0, 1\}^*$ and computes $c = (i, c_0^{(i)}, c_1^{(i)}, \pi^{(i)}) = A_2(1^k, c^*, \text{state}_1; r)$. If c is invalid with respect to pk' , that is, if $i \notin \{0, \dots, t\}$ or

$$V^{(i)}\left(\left(pk'_0, pk'_1, c_0^{(i)}, c_1^{(i)}, \text{crs}'^{(i-1)}, \dots, \text{crs}'^{(0)}\right), \pi^{(i)}, \text{crs}'^{(i)}\right) = 0$$

then S_2 outputs any ciphertext that is invalid with respect to pk (e.g., $(t+1, \perp, \perp, \perp)$). Otherwise, as in Section 4.3, S_2 utilizes the knowledge extractors guaranteed by the argument systems $\Pi^{(1)}, \dots, \Pi^{(t)}$ to generate a “certification chain” for c of the form

$$\left(c^{(0)}, f^{(0)}, \dots, c^{(i-1)}, f^{(i-1)}, c^{(i)}\right).$$

If S_2 fails in generating such a chain then it again outputs any invalid ciphertext with respect to pk . Otherwise, S_2 computes its output as follows:

1. If $c^{(0)} = c^*$ and $i = 0$, then S_2 outputs copy_1 .
2. If $c^{(0)} = c^*$ and $i > 0$, then S_2 outputs $f^{(0)} \circ \dots \circ f^{(i-1)}$.
3. If $c^{(0)} \neq c^*$, then S_2 outputs the ciphertext \tilde{c} that is obtained by “translating” c from pk' to pk as follows. First, S_2 computes $\tilde{m} = \text{Dec}_{sk'}(c^{(0)})$. Then, it computes $\tilde{c}^{(0)} = \text{Enc}_{pk}(\tilde{m})$, and $\tilde{c}^{(j)} = \text{HomEval}_{pk}(\tilde{c}^{(j-1)}, f^{(j-1)})$ for every $j \in \{1, \dots, i\}$. The ciphertext \tilde{c} is then defined as $\tilde{c}^{(i)}$.

Having described the modified simulator we now prove the following theorem stating the security of the scheme in the case $r(k) = q(k) = 1$ (noting once again that the more general case is a straightforward generalization).

Theorem 4.9. *For any constant $t \in \mathbb{N}$ and for any probabilistic polynomial-time adversary A the distributions $\{\text{Real}_{\Pi', A, t, r, q}^{\text{CCA1}}(k)\}_{k \in \mathbb{N}}$ and $\{\text{Sim}_{\Pi', S, t, r, q}^{\text{CCA1}}(k)\}_{k \in \mathbb{N}}$ are computationally indistinguishable, for $r(k) = q(k) = 1$.*

Proof. As in the proof of Theorem 4.1 we define a similar sequence of distributions $\mathcal{D}_1, \dots, \mathcal{D}_7$ such that $\mathcal{D}_1 = \text{Sim}_{\Pi', S, t, r, q}^{\text{CCA1}}$ and $\mathcal{D}_7 = \text{Real}_{\Pi', A, t, r, q}^{\text{CCA1}}$, and prove that for every $i \in \{1, \dots, 6\}$ the distributions \mathcal{D}_i and \mathcal{D}_{i+1} are computationally indistinguishable. The main difference is that in this case we change the distribution of the public key pk' chosen by the simulator, and not of the given public key pk . The proofs are very similar, and therefore here we only point out the main differences.

The distribution \mathcal{D}_1 . This is the distribution $\text{Sim}_{\Pi', S, t, r, q}^{\text{CCA1}}$.

The distribution \mathcal{D}_2 . This distribution is obtained from \mathcal{D}_1 via the following modification. As in \mathcal{D}_1 , if S_2 fails to obtain a certification chain, then output $(\text{state}_1, m, \perp)$. Otherwise, the output is computed as follows:

1. If $c^{(0)} = c^*$ and $i = 0$ then output $(\text{state}_1, m, \text{copy}_1)$. This is identical to \mathcal{D}_1 .
2. If $c^{(0)} = c^*$ and $i > 0$ then output $(\text{state}_1, m, f(m))$, where $f = f^{(0)} \circ \dots \circ f^{(i-1)}$. This is identical to \mathcal{D}_1 .
3. If $c^{(0)} \neq c^*$ then compute $m^{(0)} = \text{Dec}'_{sk'}(c^{(0)})$. If $m^{(0)} \neq \perp$ output $(\text{state}_1, m, f(m^{(0)}))$, where $f = f^{(0)} \circ \dots \circ f^{(i-1)}$, and otherwise output $(\text{state}_1, m, \perp)$. That is, in this case instead of invoking the decryption algorithm Dec' on $c^{(i)}$, we invoke it on $c^{(0)}$, and then apply the functions given by the certification chain.

The distribution \mathcal{D}_3 . This distribution is obtained from \mathcal{D}_2 by changing the distribution of pk' (that is chosen by S) and the challenge ciphertext: we produce $\text{crs}'^{(0)}$ and π^* (where $c^* = (c_0^*, c_1^*, \pi^*)$) using the simulator of the NIZK proof system $\Pi^{(0)}$.

The distribution \mathcal{D}_4 . This distribution is obtained from \mathcal{D}_3 by producing the challenge ciphertext $c^* = (c_0^*, c_1^*, \pi^*)$ with $c_0^* = \text{Enc}_{pk'_0}(m)$ (instead of $c_0^* = \text{Enc}_{pk'_0}(m')$ as in \mathcal{D}_3).

The distribution \mathcal{D}_5 . This distribution is obtained from \mathcal{D}_4 by producing the challenge ciphertext $c^* = (c_0^*, c_1^*, \pi^*)$ with $c_1^* = \text{Enc}_{pk'_1}(m)$ (instead of $c_1^* = \text{Enc}_{pk'_1}(m')$ as in \mathcal{D}_4).

The distribution \mathcal{D}_6 . This distribution is obtained from \mathcal{D}_5 by changing the distribution of pk' (that is chosen by S) and the challenge ciphertext: we produce $\text{crs}'^{(0)}$ and π^* (where $c^* = (c_0^*, c_1^*, \pi^*)$) using the algorithms $\text{CRSGen}^{(0)}$ and $\text{P}^{(0)}$, respectively (and not by using the simulator of the NIZK proof system $\Pi^{(0)}$ as in \mathcal{D}_5).

The distribution \mathcal{D}_7 . This is the distribution $\text{Real}_{\Pi', A, t, r, q}^{\text{CPA}}$.

The remainder of the proof is essentially identical to the proof of Theorem 4.1. The only subtle point is that S_1 can simulate the decryption oracle to A_1 while knowing only one of sk'_0 and sk'_1 . Specifically, given a ciphertext $(i, c_0^{(i)}, c_1^{(i)}, \pi^{(i)})$, it outputs \perp if $i \notin \{0, \dots, t\}$ or

$$\mathcal{V}^{(i)} \left((pk'_0, pk'_1, c_0^{(i)}, c_1^{(i)}, \text{crs}'^{(i-1)}, \dots, \text{crs}'^{(0)}), \pi^{(i)}, \text{crs}'^{(i)} \right) = 0 .$$

Otherwise, it computes $m_b = \text{Dec}_{sk'_b}(c_b^{(i)})$ for the value $b \in \{0, 1\}$ for which it knows the key sk'_b . The soundness of the proof system $\Pi^{(0)}$ and of the argument systems $\Pi^{(1)}, \dots, \Pi^{(t)}$ guarantee that the simulation is correct with all but a negligible probability. ■

5 The Tree-Based Construction

In this section we present our second construction which is obtained by modifying our first construction to offer a different trade-off between the length of the public key and the length of the ciphertext. As in Section 4, the scheme is parameterized by an upper bound t on the number of repeated homomorphic operations that can be applied to a ciphertext produced by the encryption algorithm. Recall that in our first construction, the length of the ciphertext is essentially independent of t , and the public key consists of $t + 1$ common reference strings. In our second construction

the number of common reference strings in the public key is only $\log t$, and a ciphertext now consists of $\log t$ ciphertexts of the underlying homomorphic scheme and $\log t$ succinct arguments. Such a trade-off may be preferable over the one offered by our first construction, for example, when using argument systems that are tailored to the \mathcal{NP} languages under considerations and or when it is not possible to use the same common reference string for all argument systems (depending, of course, on the length of the longest common reference strings).

The main idea underlying this construction is that the arguments computed by the homomorphic evaluation algorithm form a tree structure instead of a path structure. Specifically, instead of using t argument systems, we use only $d = \log t$ argument systems where the i -th one is used for arguing the well-formedness of a ciphertext after 2^i repeated homomorphic operations.

In Section 5.1 we formally specify the building blocks of the scheme, and in Section 5.2 we provide a description of the scheme and discuss its proof of security against a-priori chosen-ciphertext attacks (CCA1), which is rather similar to that of our first construction.

5.1 The Building Blocks

Our construction relies on the following building blocks:

1. A homomorphic public-key encryption scheme $\Pi = (\text{KeyGen}, \text{Enc}, \text{Dec}, \text{HomEval})$ with respect to an efficiently recognizable set of efficiently computable functions \mathcal{F} . We assume that the scheme has almost-all-key perfect decryption (see Definition 2.1). In addition, as discussed in Section 2.1, as we do not consider function privacy we assume without loss of generality that HomEval is deterministic.

For any security parameter $k \in \mathbb{N}$ we denote by $\ell_{pk} = \ell_{pk}(k)$, $\ell_m = \ell_m(k)$, $\ell_r = \ell_r(k)$, and $\ell_c = \ell_c(k)$ the bit-lengths of the public key, plaintext, randomness of Enc , and ciphertext⁷, respectively, for the scheme Π . In addition, we denote by $V_{\mathcal{F}}$ the deterministic polynomial-time algorithm for testing membership in the set \mathcal{F} , and denote by $\ell_{\mathcal{F}} = \ell_{\mathcal{F}}(k)$ the bit-length of the description of each function $f \in \mathcal{F}$.

2. A non-interactive deterministic-verifier simulation-sound adaptive zero-knowledge proof system (see Section 2.3) $\Pi^{(0)} = (\text{CRSGen}^{(0)}, \text{P}^{(0)}, \text{V}^{(0)})$ for the \mathcal{NP} -language $L^{(0)} = \bigcup_{k \in \mathbb{N}} L^{(0)}(k)$ defined as follows.

$$L^{(0)}(k) = \left\{ \left(pk_0, pk_1, c_0^{(0)}, c_1^{(0)} \right) \in \{0, 1\}^{2\ell_{pk} + 2\ell_c} : \begin{array}{l} \exists (m, r_0, r_1) \in \{0, 1\}^{\ell_m + 2\ell_r} \text{ s.t.} \\ c_0^{(0)} = \text{Enc}_{pk_0}(m; r_0) \\ \text{and } c_1^{(0)} = \text{Enc}_{pk_1}(m; r_1) \end{array} \right\}$$

For any security parameter $k \in \mathbb{N}$ we denote by $\ell_{\text{crs}^{(0)}} = \ell_{\text{crs}^{(0)}}(k)$ and $\ell_{\pi^{(0)}} = \ell_{\pi^{(0)}}(k)$ the bit-lengths of the common reference strings produced by $\text{CRSGen}^{(0)}$ and of the proofs produced by $\text{P}^{(0)}$, respectively.

3. For every $j \in \{1, \dots, d\}$ (where $d = \lceil \log t \rceil$) a $1/7$ -succinct non-interactive deterministic-verifier extractable argument system (see Section 2.2) $\Pi^{(j)} = (\text{CRSGen}^{(j)}, \text{P}^{(j)}, \text{V}^{(j)})$ for the

⁷For simplicity we assume a fixed upper bound ℓ_c on the length of ciphertexts, but this is not essential to our construction. More generally, one can allow the length of ciphertexts to increase as a result of applying the homomorphic operation.

\mathcal{NP} -language $L^{(j)} = \bigcup_{k \in \mathbb{N}} L^{(j)}(k)$ defined as follows for $j = 1$

$$L^{(1)}(k) = \left\{ \begin{array}{l} \left(pk_0, pk_1, c_0^{(1)}, c_1^{(1)}, c_0^{(2)}, c_1^{(2)} \right) \in \{0, 1\}^{2\ell_{pk} + 4\ell_c} : \\ \exists f \in \{0, 1\}^{\ell_{\mathcal{F}}} \text{ s.t.} \\ \bullet \quad \mathbf{V}_{\mathcal{F}}(f) = 1 \\ \bullet \quad c_0^{(2)} = \text{HomEval}_{pk_0} \left(c_0^{(1)}, f \right) \\ \bullet \quad c_1^{(2)} = \text{HomEval}_{pk_1} \left(c_1^{(1)}, f \right) \end{array} \right\}$$

and defined as follows for $j > 1$:

$$L^{(j)}(k) = \left\{ \begin{array}{l} \left(pk_0, pk_1, c_0^{(1)}, c_1^{(1)}, c_0^{(2^j)}, c_1^{(2^j)}, \overrightarrow{\text{crs}}^{(j-1)} \right) \in \{0, 1\}^{\ell_1^{(j)}} : \\ \exists \left(c_0^{(2^{j-1})}, c_1^{(2^{j-1})}, c_0^{(2^{j-1}+1)}, c_1^{(2^{j-1}+1)}, f, \pi_L^{(j-1)}, \pi_R^{(j-1)} \right) \in \{0, 1\}^{\ell_2^{(j)}} \text{ s.t.} \\ \bullet \quad \mathbf{V}_{\mathcal{F}}(f) = 1 \\ \bullet \quad c_0^{(2^{j-1}+1)} = \text{HomEval}_{pk_0} \left(c_0^{(2^{j-1})}, f \right) \\ \bullet \quad c_1^{(2^{j-1}+1)} = \text{HomEval}_{pk_1} \left(c_1^{(2^{j-1})}, f \right) \\ \bullet \quad \mathbf{V}^{(j-1)} \left(\left(pk_0, pk_1, c_0^{(1)}, c_1^{(1)}, c_0^{(2^{j-1})}, c_1^{(2^{j-1})}, \overrightarrow{\text{crs}}^{(j-2)} \right), \pi_L^{(j-1)}, \text{crs}^{(j-1)} \right) = 1 \\ \bullet \quad \mathbf{V}^{(j-1)} \left(\left(pk_0, pk_1, c_0^{(2^{j-1}+1)}, c_1^{(2^{j-1}+1)}, c_0^{(2^j)}, c_1^{(2^j)}, \overrightarrow{\text{crs}}^{(j-2)} \right), \pi_R^{(j-1)}, \text{crs}^{(j-1)} \right) = 1 \end{array} \right\}$$

where $\ell_1^{(j)} = 2\ell_{pk} + 4\ell_c + \sum_{t=1}^{d-1} \ell_{\text{crs}^{(t)}}$, $\ell_2^{(j)} = 4\ell_c + \ell_{\mathcal{F}} + 2\ell_{\pi^{(j-1)}}$, and for every $1 \leq j \leq d$ we define $\overrightarrow{\text{crs}}^{(j)} = (\text{crs}^{(j)}, \dots, \text{crs}^{(1)})$. For any security parameter $k \in \mathbb{N}$ we denote by $\ell_{\text{crs}^{(j)}} = \ell_{\text{crs}^{(j)}}(k)$ and $\ell_{\pi^{(j)}} = \ell_{\pi^{(j)}}(k)$ the bit-lengths of the common reference strings produced by $\text{CRSGen}^{(j)}$ and of the arguments produced by $\mathbf{P}^{(j)}$, respectively.

We note that for $j = 1$ we in fact do not need an argument system, as we can use the witness $f \in \mathcal{F}$ as the arguments. Without loss of generality we assume that $\ell_{\pi^{(1)}} \geq \max\{\ell_c, \ell_{\mathcal{F}}\}$ (as otherwise arguments can always be padded). Thus, the assumption that the argument systems $\Pi^{(2)}, \dots, \Pi^{(d)}$ are 1/7-succinct implies that the length of their arguments is upper bounded by that of $\Pi^{(1)}$ (and therefore independent of t).

5.2 The Scheme

The scheme $\Pi' = (\text{KeyGen}', \text{Enc}', \text{Dec}', \text{HomEval}')$ is parameterized by an upper bound t on the number of repeated homomorphic operations, and we let $d = \lceil \log t \rceil$. The key-generation and encryption algorithm are essentially identical to those described in Section 4.2:

- **Key generation:** On input 1^k sample two pairs of keys $(sk_0, pk_0) \leftarrow \text{KeyGen}(1^k)$ and $(sk_1, pk_1) \leftarrow \text{KeyGen}(1^k)$. Then, for every $j \in \{0, \dots, d\}$ sample $\text{crs}^{(j)} \leftarrow \text{CRSGen}^{(j)}(1^k)$. Output the secret key $sk = (sk_0, sk_1)$ and the public key $pk = (pk_0, pk_1, \text{crs}^{(0)}, \dots, \text{crs}^{(d)})$.
- **Encryption:** On input a public key pk and a plaintext m , sample $r_0, r_1 \in \{0, 1\}^*$ uniformly at random, and output the ciphertext $c^{(0)} = (c_0^{(0)}, c_1^{(0)}, \pi^{(0)})$, where

$$\begin{aligned} c_0^{(0)} &= \text{Enc}_{pk_0}(m; r_0) , \\ c_1^{(0)} &= \text{Enc}_{pk_1}(m; r_1) , \\ \pi^{(0)} &\leftarrow \mathbf{P}^{(0)} \left(\left(pk_0, pk_1, c_0^{(0)}, c_1^{(0)} \right), (m, r_0, r_1), \text{crs}^{(0)} \right) . \end{aligned}$$

- **Homomorphic evaluation:** The homomorphic evaluation algorithm follows the same approach used in Section 4.2, but computes the arguments of well-formedness in the form of a *sparse* binary tree. The leaves of the tree correspond to a chain of ciphertexts $(c^{(1)}, \dots, c^{(i)})$ that are generated from one another using the homomorphic evaluation algorithm. Each internal node at level $j \in \{1, \dots, d\}$ (where the leaves are considered to be at level 0) is a succinct argument for membership in the language $L^{(j)}$. We first describe how to generate the leaves and the internal nodes, and then describe the content of a ciphertext (i.e., which nodes of the tree should be contained in a ciphertext).

- **The leaves:** The leftmost leaf in the tree $c^{(1)} = (c_0^{(1)}, c_1^{(1)})$ is generated from a ciphertext $c^{(0)} = (c_0^{(0)}, c_1^{(0)}, \pi^{(0)})$ that is produced by the encryption algorithm and a function $f^{(0)} \in \mathcal{F}$. It is defined as

$$\begin{aligned} c_0^{(1)} &= \text{HomEval}_{pk_0} \left(c_0^{(0)}, f^{(0)} \right) , \\ c_1^{(1)} &= \text{HomEval}_{pk_1} \left(c_1^{(0)}, f^{(0)} \right) . \end{aligned}$$

From this point on both the ciphertext $c^{(0)}$, the function $f^{(0)}$, and the leaf $c^{(1)}$ are kept part of all future ciphertexts.

For every $i \in \{1, \dots, t-1\}$ the leaf $c^{(i+1)} = (c_0^{(i+1)}, c_1^{(i+1)})$ is generated from the previous leaf $c^{(i)} = (c_0^{(i)}, c_1^{(i)})$ and a function $f^{(i)} \in \mathcal{F}$ by computing

$$\begin{aligned} c_0^{(i+1)} &= \text{HomEval}_{pk_0} \left(c_0^{(i)}, f^{(i)} \right) , \\ c_1^{(i+1)} &= \text{HomEval}_{pk_1} \left(c_1^{(i)}, f^{(i)} \right) . \end{aligned}$$

- **The internal nodes:** Each internal node v at level $j \in \{1, \dots, d\}$ is an argument for membership in the language $L^{(j)}$. For $j = 1$, the two children of x are leaves $c^{(i)} = (c_0^{(i)}, c_1^{(i)})$ and $c^{(i+1)} = (c_0^{(i+1)}, c_1^{(i+1)})$, and in this case v is an argument that $c^{(i+1)}$ is obtained from $c^{(i)}$ using the homomorphic operation with some function $f^{(i)} \in \mathcal{F}$. This is computed as:

$$\pi \leftarrow \mathbf{P}^{(1)} \left((pk_0, pk_1, c_0^{(i)}, c_1^{(i)}, c_0^{(i+1)}, c_1^{(i+1)}), f^{(i)}, \text{crs}^{(1)} \right) .$$

For every $j \in \{2, \dots, d\}$, denote by v_L and v_R the two children of v . These are arguments for membership in $L^{(j-1)}$. Denote by $c^{(1)} = (c_0^{(1)}, c_1^{(1)})$ and $c^{(2^{j-1})} = (c_0^{(2^{j-1})}, c_1^{(2^{j-1})})$ the leftmost and rightmost leaves in the subtree of v_L , respectively. Similarly, denote by $c^{(2^{j-1}+1)} = (c_0^{(2^{j-1}+1)}, c_1^{(2^{j-1}+1)})$ and $c^{(2^j)} = (c_0^{(2^j)}, c_1^{(2^j)})$ the leftmost and rightmost leaves in the subtree of v_R , respectively. Then, the node v is an argument that v_L is a valid argument for $(c^{(1)}, \dots, c^{(2^{j-1})})$, v_R is a valid argument for $(c^{(2^{j-1}+1)}, \dots, c^{(2^j)})$, and that $c^{(2^{j-1}+1)}$ is obtained from $c^{(2^{j-1})}$ using the homomorphic operation with some function $f^{(2^{j-1})} \in \mathcal{F}$. This is computed as $\pi \leftarrow \mathbf{P}^{(j)}(x, w, \text{crs}^{(j)})$, where:

$$\begin{aligned} x &= (pk_0, pk_1, c_0^{(1)}, c_1^{(1)}, c_0^{(2^j)}, c_1^{(2^j)}, \overrightarrow{\text{crs}}^{(j-1)}) \\ w &= (c_0^{(2^{j-1})}, c_1^{(2^{j-1})}, c_0^{(2^{j-1}+1)}, c_1^{(2^{j-1}+1)}, f^{(2^{j-1})}, \pi_L^{(j-1)}, \pi_R^{(j-1)}) . \end{aligned}$$

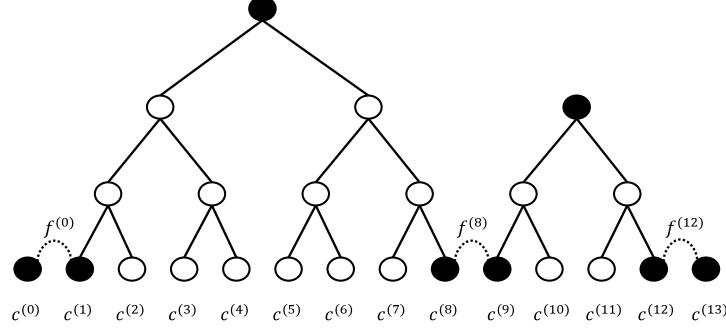


Figure 2: The above illustration shows the structure of a ciphertext after 13 repeated homomorphic operations. The ciphertext $c^{(0)}$ is an output of the encryption algorithm, and for every $i \in \{1, \dots, 13\}$ the ciphertext $c^{(i)}$ is obtained from $c^{(i-1)}$ by applying the homomorphic evaluation algorithm using a function $f^{(i)} \in \mathcal{F}$. The internal nodes on levels 1, 2, and 3 contain succinct arguments for membership in the languages $L^{(1)}$, $L^{(2)}$, and $L^{(3)}$, respectively. The ciphertext of the new scheme consists of the black nodes and the functions $f^{(0)}$, $f^{(8)}$, and $f^{(12)}$.

- **The ciphertext:** The ciphertext always includes the initial ciphertext $c^{(0)}$ that was produced by the encryption algorithm, the first leaf $c^{(1)}$, and the function $f^{(0)} \in \mathcal{F}$ that was used for generating $c^{(1)}$ from $c^{(0)}$. Then, every time we compute the value of two adjacent internal nodes v_L and v_R at some level $j - 1$ that belong to the same subtree, we compute the value of their parent v at level j , as described above. As a result, we do not longer keep any information from the subtree of v , except for its leftmost and rightmost leaves. In addition, for every two adjacent subtrees we include the function that transforms the rightmost leaf of the first subtree to the leftmost leaf of the second subtree. Note that such subtrees must be of different depths, as otherwise they are merged. Thus, at any point in time a ciphertext may contain at most $2d + 1$ ciphertexts of the underlying scheme, d short arguments, and d descriptions of functions from \mathcal{F} (connecting subtrees). See Figure 2 for an illustration of the structure of a ciphertext.
- **Decryption:** On input the secret key sk and a ciphertext of the above form, verify the validity of the non-interactive zero-knowledge proof contained in $c^{(0)}$, verify that $c^{(1)}$ is obtained from $c^{(0)}$ using the function $f^{(0)} \in \mathcal{F}$, verify that the given tree has the right structure (with functions from \mathcal{F} connecting subtrees), and verify the validity of all the arguments in the non-empty internal nodes of the tree. If any of these verifications fail, then output \perp . Otherwise, compute $m_0 = \text{Dec}_{sk_0}(c_0^{(i)})$ and $m_1 = \text{Dec}_{sk_1}(c_1^{(i)})$, where $c^{(i)} = (c_0^{(i)}, c_1^{(i)})$ is the rightmost leaf. If $m_0 \neq m_1$ then output \perp , and otherwise output m_0 .

Chosen-ciphertext security. As this scheme is obtained from the one in Section 4 by only changing the structure of the arguments that generate the ciphertext, the proof of security is rather similar to that in Sections 4.3 and 4.4. The only difference is in the way S_2 produces the “certification chain” for the ciphertext that the adversary outputs: instead of using the path structure of the ciphertext, the simulator now uses the tree structure of the ciphertext and applies the knowledge extractors accordingly. The remainder of the proof is exactly the same.

6 Extensions and Open Problems

We conclude the paper with a discussion of several extensions of our work and open problems.

The number of repeated homomorphic operations. Our schemes allow any pre-specified constant bound $t \in \mathbb{N}$ on the number of repeated homomorphic operations. It would be interesting to allow this bound to be a function $t(k)$ of the security parameter. As discussed in Section 2.2, the bottleneck is the super-linear length of the common-reference string in Groth’s and Lipmaa’s argument systems [Gro10, Lip11]. Any improvement to these argument systems with a common-reference string of linear length will directly allow any logarithmic number of repeated homomorphic operations in the path-based scheme, and any polynomial number of such operations in the tree-based scheme.

Function privacy and unlinkability. For some applications a homomorphic encryption scheme may be required to ensure *function privacy* [GHV10] or even *unlinkability* [PR08]. Function privacy asks that the homomorphic evaluation algorithm does not reveal (in a semantic security fashion) which operation it applies, and unlinkability asks that the output of the homomorphic evaluation algorithm is computationally indistinguishable from the output of the encryption algorithm. For example, the voting application discussed in the introduction requires function privacy to ensure that individual votes remain private. Our approach in this paper focuses on preventing a blow-up in the length of ciphertexts, and incorporating function privacy and unlinkability into our framework is an interesting direction for future work. We note that since Groth’s argument system [Gro10] is also zero-knowledge it is quite plausible to show that ciphertexts in our schemes reveal nothing more than the number of repeated homomorphic operations.

A-posteriori chosen-ciphertext security (CCA2). As discussed in Section 1.3, Prabhakaran and Rosulek [PR08] considered the rather orthogonal problem of providing a homomorphic encryption scheme that is secure against a meaningful variant of a-posteriori chosen-ciphertext attacks (CCA2). In light of the fact that our schemes already offer targeted malleability against a-priori chosen-ciphertext attacks (CCA1), it would be interesting to extend our approach to the setting considered by Prabhakaran and Rosulek.

References

- [ADR02] J. H. An, Y. Dodis, and T. Rabin. On the security of joint signature and encryption. In *Advances in Cryptology – EUROCRYPT ’02*, pages 83–107, 2002.
- [AIK10] B. Applebaum, Y. Ishai, and E. Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. In *Proceedings of the 37th International Colloquium on Automata, Languages and Programming*, pages 152–163, 2010.
- [BFM88] M. Blum, P. Feldman, and S. Micali. Non-interactive zero-knowledge and its applications. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, pages 103–112, 1988.
- [BP04a] M. Bellare and A. Palacio. The knowledge-of-exponent assumptions and 3-round zero-knowledge protocols. In *Advances in Cryptology – CRYPTO ’04*, pages 273–289, 2004.
- [BP04b] M. Bellare and A. Palacio. Towards plaintext-aware public-key encryption without random oracles. In *Advances in Cryptology – ASIACRYPT ’04*, pages 48–62, 2004.
- [BR93] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

- [BS99] M. Bellare and A. Sahai. Non-malleable encryption: Equivalence between two notions, and an indistinguishability-based characterization. In *Advances in Cryptology – CRYPTO ’99*, pages 519–536, 1999. The full version is available as Cryptology ePrint Archive, Report 2006/228.
- [BSM⁺91] M. Blum, A. D. Santis, S. Micali, and G. Persiano. Noninteractive zero-knowledge. *SIAM Journal on Computing*, 20(6):1084–1118, 1991.
- [CGS97] R. Cramer, R. Gennaro, and B. Schoenmakers. A secure and optimally efficient multi-authority election scheme. In *Advances in Cryptology – EUROCRYPT ’97*, pages 103–118, 1997.
- [CKN03] R. Canetti, H. Krawczyk, and J. B. Nielsen. Relaxing chosen-ciphertext security. In *Advances in Cryptology – CRYPTO ’03*, pages 565–582, 2003.
- [CKV10] K.-M. Chung, Y. Kalai, and S. Vadhan. Improved delegation of computation using fully homomorphic encryption. In *Advances in Cryptology – CRYPTO ’10*, pages 483–501, 2010.
- [CS98] R. Cramer and V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *Advances in Cryptology – CRYPTO ’98*, pages 13–25, 1998.
- [CT10] A. Chiesa and E. Tromer. Proof-carrying data and hearsay arguments from signature cards. In *Proceedings of the 1st Symposium on Innovations in Computer Science*, pages 310–331, 2010.
- [Dam91] I. Damgård. Towards practical public key systems secure against chosen ciphertext attacks. In *Advances in Cryptology – CRYPTO ’91*, pages 445–456, 1991.
- [DDN00] D. Dolev, C. Dwork, and M. Naor. Non-malleable cryptography. *SIAM Journal on Computing*, 30(2):391–437, 2000.
- [DNR04] C. Dwork, M. Naor, and O. Reingold. Immunizing encryption schemes from decryption errors. In *Advances in Cryptology – EUROCRYPT ’04*, pages 342–360, 2004.
- [FLS90] U. Feige, D. Lapidot, and A. Shamir. Multiple non-interactive zero knowledge proofs based on a single random string. In *Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science*, pages 308–317, 1990.
- [Gen09] C. Gentry. A fully homomorphic encryption scheme. PhD Thesis, Stanford University, 2009. Available at <http://crypto.stanford.edu/craig>.
- [GGP10] R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *Advances in Cryptology – CRYPTO ’10*, pages 465–482, 2010.
- [GHV10] C. Gentry, S. Halevi, and V. Vaikuntanathan. *i*-hop homomorphic encryption and rerandomizable Yao circuits. In *Advances in Cryptology – CRYPTO ’10*, pages 155–172, 2010.
- [GKR08] S. Goldwasser, Y. T. Kalai, and G. Rothblum. Delegating computation: Interactive proofs for muggles. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing*, pages 113–122, 2008.

- [Gro04] J. Groth. Rerandomizable and replayable adaptive chosen ciphertext attack secure cryptosystems. In *Proceedings of the 1st Theory of Cryptography Conference*, pages 152–170, 2004.
- [Gro10] J. Groth. Short pairing-based non-interactive zero-knowledge arguments. In *Advances in Cryptology – ASIACRYPT ’10*, pages 321–340, 2010.
- [GW11] C. Gentry and D. Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *Proceedings of the 43rd Annual ACM Symposium on Theory of Computing*, pages 99–108, 2011.
- [Lin06] Y. Lindell. A simpler construction of CCA2-secure public-key encryption under general assumptions. *Journal of Cryptology*, 19(3):359–377, 2006.
- [Lip11] H. Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. Cryptology ePrint Archive, Report 2011/009, 2011.
- [Mic00] S. Micali. Computationally sound proofs. *SIAM Journal of Computing*, 30(4):1253–1298, 2000. An extended abstract appeared in *Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science*, 1994.
- [Nao03] M. Naor. On cryptographic assumptions and challenges. In *Advances in Cryptology – CRYPTO ’03*, pages 96–109, 2003.
- [NY90] M. Naor and M. Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, pages 427–437, 1990.
- [PR07] M. Prabhakaran and M. Rosulek. Rerandomizable RCCA encryption. In *Advances in Cryptology – CRYPTO ’07*, pages 517–534, 2007.
- [PR08] M. Prabhakaran and M. Rosulek. Homomorphic encryption with CCA security. In *Proceedings of the 35th International Colloquium on Automata, Languages and Programming*, pages 667–678, 2008.
- [PSV07] R. Pass, A. Shelat, and V. Vaikuntanathan. Relations among notions of non-malleability for encryption. In *Advances in Cryptology – ASIACRYPT ’07*, pages 519–535, 2007.
- [RAD78] R. Rivest, L. Adleman, and M. Dertouzos. On data banks and privacy homomorphisms. *Foundations of Secure Computation*, 1978.
- [Sah99] A. Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science*, pages 543–553, 1999.
- [SV10] N. P. Smart and F. Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In *Public Key Cryptography – PKC ’10*, pages 420–443, 2010.
- [Val08] P. Valiant. Incrementally verifiable computation – or – proofs of knowledge imply time/space efficiency. In *Proceedings of the 5th Theory of Cryptography Conference*, pages 1–18, 2008.

- [vDGH⁺10] M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan. Fully homomorphic encryption over the integers. In *Advances in Cryptology – EUROCRYPT ’10*, pages 24–43, 2010.

Trapdoors for Lattices: Simpler, Tighter, Faster, Smaller

Daniele Micciancio*

Chris Peikert[†]

September 14, 2011

Abstract

We give new methods for generating and using “strong trapdoors” in cryptographic lattices, which are simultaneously simple, efficient, easy to implement (even in parallel), and asymptotically optimal with very small hidden constants. Our methods involve a new kind of trapdoor, and include specialized algorithms for inverting LWE, randomly sampling SIS preimages, and securely delegating trapdoors. These tasks were previously the main bottleneck for a wide range of cryptographic schemes, and our techniques substantially improve upon the prior ones, both in terms of practical performance and quality of the produced outputs. Moreover, the simple structure of the new trapdoor and associated algorithms can be exposed in applications, leading to further simplifications and efficiency improvements. We exemplify the applicability of our methods with new digital signature schemes and CCA-secure encryption schemes, which have better efficiency and security than the previously known lattice-based constructions.

1 Introduction

Cryptography based on lattices has several attractive and distinguishing features:

- On the *security* front, the best attacks on the underlying problems require exponential $2^{\Omega(n)}$ time in the main security parameter n , even for quantum adversaries. By contrast, for example, mainstream factoring-based cryptography can be broken in subexponential $2^{\tilde{O}(n^{1/3})}$ time classically, and even in polynomial $n^{O(1)}$ time using quantum algorithms. Moreover, lattice cryptography is supported by strong worst-case/average-case security reductions, which provide solid theoretical evidence that the random instances used in cryptography are indeed asymptotically hard, and do not suffer from any unforeseen “structural” weaknesses.

*University of California, San Diego. Email: textttddaniele@cs.ucsd.edu. This material is based on research sponsored by DARPA under agreement number FA8750-11-C-0096. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the U.S. Government.

[†]School of Computer Science, College of Computing, Georgia Institute of Technology. Email: cpeikert@cc.gatech.edu. This material is based upon work supported by the National Science Foundation under Grant CNS-0716786 and CAREER Award CCF-1054495, by the Alfred P. Sloan Foundation, and by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under Contract No. FA8750-11-C-0098. The views expressed are those of the authors and do not necessarily reflect the official policy or position of the National Science Foundation, the Sloan Foundation, DARPA or the U.S. Government.

- On the *efficiency* and *implementation* fronts, lattice cryptography operations can be extremely simple, fast and parallelizable. Typical operations are the selection of uniformly random integer matrices \mathbf{A} modulo some small $q = \text{poly}(n)$, and the evaluation of simple linear functions like

$$f_{\mathbf{A}}(\mathbf{x}) := \mathbf{A}\mathbf{x} \bmod q \quad \text{and} \quad g_{\mathbf{A}}(\mathbf{s}, \mathbf{e}) := \mathbf{s}^t \mathbf{A} + \mathbf{e}^t \bmod q$$

on short integer vectors \mathbf{x}, \mathbf{e} .¹ (For commonly used parameters, $f_{\mathbf{A}}$ is surjective while $g_{\mathbf{A}}$ is injective.) Often, the modulus q is small enough that all the basic operations can be directly implemented using machine-level arithmetic. By contrast, the analogous operations in number-theoretic cryptography (e.g., generating huge random primes, and exponentiating modulo such primes) are much more complex, admit only limited parallelism in practice, and require the use of “big number” arithmetic libraries.

In recent years lattice-based cryptography has also been shown to be extremely versatile, leading to a large number of theoretical applications ranging from (hierarchical) identity-based encryption [GPV08, CHKP10, ABB10a, ABB10b], to fully homomorphic encryption schemes [Gen09b, Gen09a, vGHV10, BV11b, BV11a, GH11, BGV11], and much more (e.g., [LM08, PW08, Lyu08, PV08, PVW08, Pei09b, ACPS09, Rüc10, Boy10, GHV10, GKV10]).

Not all lattice cryptography is as simple as selecting random matrices \mathbf{A} and evaluating linear functions like $f_{\mathbf{A}}(\mathbf{x}) = \mathbf{A}\mathbf{x} \bmod q$, however. In fact, such operations yield only collision-resistant hash functions, public-key encryption schemes that are secure under passive attacks, and little else. Richer and more advanced lattice-based cryptographic schemes, including chosen ciphertext-secure encryption, “hash-and-sign” digital signatures, and identity-based encryption also require generating a matrix \mathbf{A} together with some “*strong*” *trapdoor*, typically in the form of a nonsingular square matrix (a basis) \mathbf{S} of short integer vectors such that $\mathbf{A}\mathbf{S} = \mathbf{0} \bmod q$. (The matrix \mathbf{S} is usually interpreted as a basis of a lattice defined by using \mathbf{A} as a “parity check” matrix.) Applications of such strong trapdoors also require certain efficient inversion algorithms for the functions $f_{\mathbf{A}}$ and $g_{\mathbf{A}}$, using \mathbf{S} . Appropriately inverting $f_{\mathbf{A}}$ can be particularly complex, as it typically requires sampling *random preimages* of $f_{\mathbf{A}}(\mathbf{x})$ according to a Gaussian-like probability distribution (see [GPV08]).

Theoretical solutions for all the above tasks (generating \mathbf{A} with strong trapdoor \mathbf{S} [Ajt99, AP09], trapdoor inversion of $g_{\mathbf{A}}$ and preimage sampling for $f_{\mathbf{A}}$ [GPV08]) are known, but they are rather complex and not very suitable for practice, in either runtime or the “quality” of their outputs. (The quality of a trapdoor \mathbf{S} roughly corresponds to the Euclidean lengths of its vectors — shorter is better.) The current best method for trapdoor generation [AP09] is conceptually and algorithmically complex, and involves costly computations of Hermite normal forms and matrix inverses. And while the dimensions and quality of its output are *asymptotically* optimal (or nearly so, depending on the precise notion of quality), the hidden constant factors are rather large. Similarly, the standard methods for inverting $g_{\mathbf{A}}$ and sampling preimages of $f_{\mathbf{A}}$ [Bab85, Kle00, GPV08] are inherently sequential and time-consuming, as they are based on an orthogonalization process that uses high-precision real numbers. A more efficient and parallelizable method for preimage sampling (which uses only small-integer arithmetic) has recently been discovered [Pei10], but it is still more complex than is desirable for practice, and the quality of its output can be slightly worse than that of the sequential algorithm when using the same trapdoor \mathbf{S} .

More compact and efficient trapdoors appear necessary for bringing advanced lattice-based schemes to practice, not only because of the current unsatisfactory runtimes, but also because the concrete security of lattice cryptography can be quite sensitive to even small changes in the main parameters. As already

¹ Inverting these functions corresponds to solving the “short integer solution” (SIS) problem [Ajt96] for $f_{\mathbf{A}}$, and the “learning with errors” (LWE) problem [Reg05] for $g_{\mathbf{A}}$, both of which are widely used in lattice cryptography and enjoy provable worst-case hardness.

mentioned, two central objects are a uniformly random matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ that serves as a public key, and an associated secret matrix $\mathbf{S} \in \mathbb{Z}^{m \times m}$ consisting of short integer vectors having “quality” s , where smaller is better. Here n is the main security parameter governing the hardness of breaking the functions, and m is the dimension of a lattice associated with \mathbf{A} , which is generated by the vectors in \mathbf{S} . Note that the security parameter n and lattice dimension m need not be the same; indeed, typically we have $m = \Theta(n \lg q)$, which for many applications is optimal up to constant factors. (For simplicity, throughout this introduction we use the base-2 logarithm; other choices are possible and yield tradeoffs among the parameters.) For the trapdoor quality, achieving $s = O(\sqrt{m})$ is asymptotically optimal, and random preimages of $f_{\mathbf{A}}$ generated using \mathbf{S} have Euclidean length $\beta \approx s\sqrt{m}$. For security, it must be hard (*without* knowing the trapdoor) to find any preimage having length bounded by β . Interestingly, the computational resources needed to do so can increase dramatically with only a moderate decrease in the bound β (see, e.g., [GN08, MR09]). Therefore, improving the parameters m and s by even small constant factors can have a significant impact on concrete security. Moreover, this can lead to a “virtuous cycle” in which the increased security allows for the use of a smaller security parameter n , which leads to even smaller values of m , s , and β , etc. Note also that the schemes’ key sizes and concrete runtimes are reduced as well, so improving the parameters yields a “win-win-win” scenario of simultaneously smaller keys, increased concrete security, and faster operations. (This phenomenon is borne out concretely; see Figure 2.)

1.1 Contributions

The first main contribution of this paper is a new method of trapdoor generation for cryptographic lattices, which is simultaneously simple, efficient, easy to implement (even in parallel), and asymptotically optimal with small hidden constants. The new trapdoor generator strictly subsumes the prior ones of [Ajt99, AP09], in that it proves the main theorems from those works, but with improved concrete bounds for all the relevant quantities (simultaneously), and via a conceptually simpler and more efficient algorithm. To accompany our trapdoor generator, we also give specialized algorithms for trapdoor inversion (for $g_{\mathbf{A}}$) and preimage sampling (for $f_{\mathbf{A}}$), which are simpler and more efficient in our setting than the prior general solutions [Bab85, Kle00, GPV08, Pei10].

Our methods yield large constant-factor improvements, and in some cases even small asymptotic improvements, in the lattice dimension m , trapdoor quality s , and storage size of the trapdoor. Because trapdoor generation and inversion algorithms are the main operations in many lattice cryptography schemes, our algorithms can be plugged in as ‘black boxes’ to deliver significant concrete improvements in all such applications. Moreover, it is often possible to expose the special (and very simple) structure of our trapdoor directly in cryptographic schemes, yielding additional improvements and potentially new applications. (Below we summarize a few improvements to existing applications, with full details in Section 6.)

We now give a detailed comparison of our results with the most relevant prior works [Ajt99, AP09, GPV08, Pei10]. The quantitative improvements are summarized in Figure 1.

Simpler, faster trapdoor generation and inversion algorithms. Our trapdoor generator is exceedingly simple, especially as compared with the prior constructions [Ajt99, AP09]. It essentially amounts to just one multiplication of two random matrices, whose entries are chosen independently from appropriate probability distributions. Surprisingly, this method is nearly identical to Ajtai’s original method [Ajt96] of generating a random lattice together with a “weak” trapdoor of one or more short vectors (but *not* a full basis), with one added twist. And while there are no detailed runtime analyses or public implementations of [Ajt99, AP09], it is clear from inspection that our new method is significantly more efficient, since it does not involve any expensive Hermite normal form or matrix inversion computations.

Our specialized, parallel inversion algorithms for $f_{\mathbf{A}}$ and $g_{\mathbf{A}}$ are also simpler and more practically efficient than the general solutions of [Bab85, Kle00, GPV08, Pei10] (though we note that our trapdoor generator is entirely compatible with those general algorithms as well). In particular, we give the first *parallel* algorithm for inverting $g_{\mathbf{A}}$ under asymptotically optimal error rates (previously, handling such large errors required the sequential “nearest-plane” algorithm of [Bab85]), and our preimage sampling algorithm for $f_{\mathbf{A}}$ works with smaller integers and requires much less offline storage than the one from [Pei10].

Tighter parameters. To generate a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ that is within negligible statistical distance of uniform, our new trapdoor construction improves the lattice dimension from $m > 5n \lg q$ [AP09] down to $m \approx 2n \lg q$. (In both cases, the base of the logarithm is a tunable parameter that appears as a multiplicative factor in the quality of the trapdoor; here we fix upon base 2 for concreteness.) In addition, we give the first known *computationally pseudorandom* construction (under the LWE assumption), where the dimension can be as small as $m = n(1 + \lg q)$, although at the cost of an $\Omega(\sqrt{n})$ factor worse quality s .

Our construction also greatly improves the quality s of the trapdoor. The best prior construction [AP09] produces a basis whose Gram-Schmidt quality (i.e., the maximum length of its Gram-Schmidt orthogonalized vectors) was loosely bounded by $20\sqrt{n \lg q}$. However, the Gram-Schmidt notion of quality is useful only for less efficient, sequential inversion algorithms [Bab85, GPV08] that use high-precision real arithmetic. For the more efficient, parallel preimage sampling algorithm of [Pei10] that uses small-integer arithmetic, the parameters guaranteed by [AP09] are asymptotically worse, at $m > n \lg^2 q$ and $s \geq 16\sqrt{n \lg^2 q}$. By contrast, our (statistically secure) trapdoor construction achieves the “best of both worlds:” asymptotically optimal dimension $m \approx 2n \lg q$ and quality $s \approx 1.6\sqrt{n \lg q}$ or better, with a parallel preimage sampling algorithm that is slightly more efficient than the one of [Pei10].

Altogether, for any n and typical values of $q \geq 2^{16}$, we conservatively estimate that the new trapdoor generator and inversion algorithms collectively provide at least a $7 \lg q \geq 112$ -fold *improvement* in the length bound $\beta \approx s\sqrt{m}$ for $f_{\mathbf{A}}$ preimages (generated using an efficient algorithm). We also obtain similar improvements in the size of the error terms that can be handled when efficiently inverting $g_{\mathbf{A}}$.

New, smaller trapdoors. As an additional benefit, our construction actually produces a *new kind of trapdoor* — not a basis — that is at least 4 times smaller in storage than a basis of corresponding quality, and is at least as powerful, i.e., a good basis can be efficiently derived from the new trapdoor. We stress that our specialized inversion algorithms using the new trapdoor provide almost exactly the same quality as the inefficient, sequential algorithms using a derived basis, so there is no trade-off between efficiency and quality. (This is in contrast with [Pei10] when using a basis generated according to [AP09].) Moreover, the storage size of the new trapdoor grows only linearly in the lattice dimension m , rather than quadratically as a basis does. This is most significant for applications like hierarchical ID-based encryption [CHKP10, ABB10a] that *delegate* trapdoors for increasing values of m . The new trapdoor also admits a very simple and efficient delegation mechanism, which unlike the prior method [CHKP10] does not require any costly operations like linear independence tests, or conversions from a full-rank set of lattice vectors into a basis. In summary, the new type of trapdoor and its associated algorithms are *strictly preferable* to a short basis in terms of algorithmic efficiency, output quality, and storage size (simultaneously).

Ring-based constructions. Finally, and most importantly for practice, all of the above-described constructions and algorithms extend immediately to the *ring* setting, where functions analogous to $f_{\mathbf{A}}$ and $g_{\mathbf{A}}$ require only quasi-linear $\tilde{O}(n)$ space and time to specify and evaluate (respectively), which is a factor of $\tilde{\Omega}(n)$ improvement over the matrix-based functions defined above. See the representative works [Mic02, PR06, LM06, LMPR08, LPR10] for more details on these functions and their security foundations.

	[Ajt99, AP09] constructions	This work (fast $f_{\mathbf{A}}^{-1}$)	Factor Improvement
Dimension m	slow $f_{\mathbf{A}}^{-1}$ [Kle00, GPV08]: $> 5n \lg q$ fast $f_{\mathbf{A}}^{-1}$ [Pei10]: $> n \lg^2 q$	$2n \lg q$ ($\overset{s}{\approx}$) $n(1 + \lg q)$ ($\overset{c}{\approx}$)	$2.5 - \lg q$
Quality s	slow $f_{\mathbf{A}}^{-1}$: $\approx 20\sqrt{n \lg q}$ fast $f_{\mathbf{A}}^{-1}$: $\approx 16\sqrt{n \lg^2 q}$	$\approx 1.6\sqrt{n \lg q}$ ($\overset{s}{\approx}$)	$12.5 - 10\sqrt{\lg q}$
Length $\beta \approx s\sqrt{m}$	slow $f_{\mathbf{A}}^{-1}$: $> 45n \lg q$ fast $f_{\mathbf{A}}^{-1}$: $> 16n \lg^2 q$	$\approx 2.3n \lg q$ ($\overset{s}{\approx}$)	$19 - 7 \lg q$

Figure 1: Summary of parameters for our constructions and algorithms versus prior ones. In the column labelled “this work,” $\overset{s}{\approx}$ and $\overset{c}{\approx}$ denote constructions producing public keys \mathbf{A} that are statistically close to uniform, and computationally pseudorandom, respectively. (All quality terms s and length bounds β omit the same statistical “smoothing” factor for \mathbb{Z} , which is about 4–5 in practice.)

To illustrate the kinds of concrete improvements that our methods provide, in Figure 2 we give representative parameters for the canonical application of GPV signatures [GPV08], comparing the old and new trapdoor constructions for nearly equal levels of concrete security. We stress that these parameters are not highly optimized, and making adjustments to some of the tunable parameters in our constructions may provide better combinations of efficiency and concrete security. We leave this effort for future work.

1.2 Techniques

The main idea behind our new method of trapdoor generation is as follows. Instead of building a random matrix \mathbf{A} through some specialized and complex process, we start from a carefully crafted *public* matrix \mathbf{G} (and its associated lattice), for which the associated functions $f_{\mathbf{G}}$ and $g_{\mathbf{G}}$ admit very efficient (in both sequential and parallel complexity) and high-quality inversion algorithms. In particular, preimage sampling for $f_{\mathbf{G}}$ and inversion for $g_{\mathbf{G}}$ can be performed in essentially $O(n \log n)$ sequential time, and can even be performed by n parallel $O(\log n)$ -time operations or table lookups. (This should be compared with the general algorithms for these tasks, which require at least quadratic $\Omega(n^2 \log^2 n)$ time, and are not always parallelizable for optimal noise parameters.) We emphasize that \mathbf{G} is *not* a cryptographic key, but rather a fixed and public matrix that may be used by all parties, so the implementation of all its associated operations can be highly optimized, in both software and hardware. We also mention that the simplest and most practically efficient choices of \mathbf{G} work for a modulus q that is a power of a small prime, such as $q = 2^k$, but a crucial search/decision reduction for LWE was not previously known for such q , despite its obvious practical utility. In Section 3 we provide a very general reduction that covers this case and others, and subsumes all of the known (and incomparable) search/decision reductions for LWE [BFKL93, Reg05, Pei09b, ACPS09].

To generate a *random* matrix \mathbf{A} with a trapdoor, we take two additional steps: first, we extend \mathbf{G} into a semi-random matrix $\mathbf{A}' = [\bar{\mathbf{A}} \mid \mathbf{G}]$, for uniform $\bar{\mathbf{A}} \in \mathbb{Z}_q^{n \times \bar{m}}$ and sufficiently large \bar{m} . (As shown in [CHKP10], inversion of $g_{\mathbf{A}'}$ and preimage sampling for $f_{\mathbf{A}'}$ reduce very efficiently to the corresponding tasks for $g_{\mathbf{G}}$ and $f_{\mathbf{G}}$.) Finally, we simply apply to \mathbf{A}' a certain random unimodular transformation defined by the matrix $\mathbf{T} = \begin{bmatrix} \mathbf{I} & -\mathbf{R} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}$, for a random “short” secret matrix \mathbf{R} that will serve as the trapdoor, to obtain

$$\mathbf{A} = \mathbf{A}' \cdot \mathbf{T} = [\bar{\mathbf{A}} \mid \mathbf{G} - \bar{\mathbf{A}}\mathbf{R}].$$

	[AP09] with fast f_A^{-1}	This work	Factor Improvement
Sec param n	436	284	1.5
Modulus q	2^{32}	2^{24}	256
Dimension m	446,644	13,812	32.3
Quality s	10.7×10^3	418	25.6
Length β	12.9×10^6	91.6×10^3	141
Key size (bits)	6.22×10^9	92.2×10^6	67.5
Key size (ring-based)	$\approx 16 \times 10^6$	$\approx 361 \times 10^3$	\approx 44.3

Figure 2: Representative parameters for GPV signatures (using fast inversion algorithms) for the old and new trapdoor generation methods. Using the methodology from [MR09], both sets of parameters have security level corresponding to a parameter δ of at most 1.007, which is estimated to require about 2^{46} core-years on a 64-bit 1.86GHz Xeon using the state-of-the-art in lattice basis reduction [GN08, CN11]. We use a smoothing parameter of $r = 4.5$ for \mathbb{Z} , which corresponds to statistical error of less than 2^{-90} for each randomized-rounding operation during signing. Key sizes are calculated using the Hermite normal form optimization. Key sizes for *ring-based* GPV signatures are approximated to be smaller by a factor of about $0.9n$.

The transformation given by \mathbf{T} has the following properties:

- It is very easy to compute and invert, requiring essentially just one multiplication by \mathbf{R} in both cases. (Note that $\mathbf{T}^{-1} = \begin{bmatrix} \mathbf{I} & \mathbf{R} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}$.)
- It results in a matrix \mathbf{A} that is distributed essentially uniformly at random, as required by the security reductions (and worst-case hardness proofs) for lattice-based cryptographic schemes.
- For the resulting functions f_A and g_A , preimage sampling and inversion very simply and efficiently reduce to the corresponding tasks for f_G , g_G . The overhead of the reduction is essentially just a single matrix-vector product with the secret matrix \mathbf{R} (which, when inverting f_A , can largely be precomputed even before the target value is known).

As a result, the cost of the inversion operations ends up being very close to that of computing f_A and g_A in the forward direction. Moreover, the fact that the running time is dominated by matrix-vector multiplications with the *fixed* trapdoor matrix \mathbf{R} yields theoretical (but asymptotically significant) improvements in the context of batch execution of several operations relative to the same secret key \mathbf{R} : instead of evaluating several products $\mathbf{R}\mathbf{z}_1, \mathbf{R}\mathbf{z}_2, \dots, \mathbf{R}\mathbf{z}_n$ individually at a total cost of $\Omega(n^3)$, one can employ fast matrix multiplication techniques to evaluate $\mathbf{R}[\mathbf{z}_1, \dots, \mathbf{z}_n]$ as a whole in subcubic time. Batch operations can be exploited in applications like the multi-bit IBE of [GPV08] and its extensions to HIBE [CHKP10, ABB10a, ABB10b].

Related techniques. At the surface, our trapdoor generator appears similar to the original “GGH” approach of [GGH97] for generating a lattice together with a short basis. That technique works by choosing some random short vectors as the secret “good basis” of a lattice, and then transforms them into a public “bad basis” for the *same* lattice, via a unimodular matrix having large entries. (Note, though, that this does not produce a lattice from Ajtai’s worst-case-hard family.) A closer look reveals, however, that (worst-case hardness aside) our method is actually *not* an instance of the GGH paradigm: here the initial short basis of the lattice

defined by \mathbf{G} (or the semi-random matrix $[\bar{\mathbf{A}}|\mathbf{G}]$) is *fixed* and *public*, while the random unimodular matrix $\mathbf{T} = \begin{bmatrix} \mathbf{I} & -\mathbf{R} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}$ actually *produces a new lattice* by applying a (reversible) linear transformation to the original lattice. In other words, in contrast with GGH we multiply a (short) unimodular matrix on the “other side” of the original short basis, thus changing the lattice it generates.

A more appropriate comparison is to Ajtai’s original method [Ajt96] for generating a random \mathbf{A} together with a “weak” trapdoor of one or more short lattice vectors (but *not* a full basis). There, one simply chooses a semi-random matrix $\mathbf{A}' = [\bar{\mathbf{A}} | \mathbf{0}]$ and outputs $\mathbf{A} = \mathbf{A}' \cdot \mathbf{T} = [\bar{\mathbf{A}} | -\bar{\mathbf{A}}\mathbf{R}]$, with short vectors $\begin{bmatrix} \mathbf{R} \\ \mathbf{I} \end{bmatrix}$. Perhaps surprisingly, our strong trapdoor generator is just a simple twist on Ajtai’s original weak generator, replacing $\mathbf{0}$ with the gadget \mathbf{G} .

Our constructions and inversion algorithms also draw upon several other techniques from throughout the literature. The trapdoor basis generator of [AP09] and the LWE-based “lossy” injective trapdoor function of [PW08] both use a fixed “gadget” matrix analogous to \mathbf{G} , whose entries grow geometrically in a structured way. In both cases, the gadget is concealed (either statistically or computationally) in the public key by a small combination of uniformly random vectors. Our method for adding tags to the trapdoor is very similar to a technique for doing the same with the lossy TDF of [PW08], and is identical to the method used in [ABB10a] for constructing compact (H)IBE. Finally, in our preimage sampling algorithm for $f_{\mathbf{A}}$, we use the “convolution” technique from [Pei10] to correct for some statistical skew that arises when converting preimages for $f_{\mathbf{G}}$ to preimages for $f_{\mathbf{A}}$, which would otherwise leak information about the trapdoor \mathbf{R} .

1.3 Applications

Our improved trapdoor generator and inversion algorithms can be plugged into any scheme that uses such tools as a “black box,” and the resulting scheme will inherit all the efficiency improvements. (Every application we know of admits such a black-box replacement.) Moreover, the special properties of our methods allow for further improvements to the design, efficiency, and security reductions of existing schemes. Here we summarize some representative improvements that are possible to obtain; see Section 6 for complete details.

Hash-and-sign digital signatures. Our construction and supporting algorithms plug directly into the “full domain hash” signature scheme of [GPV08], which is strongly unforgeable in the random oracle model, with a tight security reduction. One can even use our computationally secure trapdoor generator to obtain a smaller public verification key, though at the cost of a hardness-of-LWE assumption, and a somewhat stronger SIS assumption (which affects concrete security). Determining the right balance between key size and security is left for later work.

In the standard model, there are two closely related types of hash-and-sign signature schemes:

- The one of [CHKP10], which has signatures of bit length $\tilde{O}(n^2)$, and is existentially unforgeable (later improved to be strongly unforgeable [Rüc10]) assuming the hardness of inverting $f_{\mathbf{A}}$ with solution length bounded by $\beta = \tilde{O}(n^{1.5})$.²
- The scheme of [Boy10], a lattice analogue of the pairing-based signature of [Wat05], which has signatures of bit length $\tilde{O}(n)$ and is existentially unforgeable assuming the hardness of inverting $f_{\mathbf{A}}$ with solution length bounded by $\beta = \tilde{O}(n^{3.5})$.

We improve the latter scheme in several ways, by: (i) improving the length bound to $\beta = \tilde{O}(n^{2.5})$; (ii) reducing the online runtime of the signing algorithm from $\tilde{O}(n^3)$ to $\tilde{O}(n^2)$ via chameleon hashing [KR00]; (iii) making the scheme strongly unforgeable *a la* [GPV08, Rüc10]; (iv) giving a tighter and simpler security reduction

²All parameters in this discussion assume a message length of $\tilde{O}(n)$ bits.

(using a variant of the “prefix technique” [HW09] as in [CHKP10]), where the reduction’s advantage degrades only linearly in the number of signature queries; and (v) removing all additional constraints on the parameters n and q (aside from those needed to ensure hardness of the SIS problem). We stress that the scheme itself is essentially the same (up to the improved and generalized parameters, and chameleon hashing) as that of [Boy10]; only the security proof and underlying assumption are improved. Note that in comparison with [CHKP10], there is still a trade-off between the bit length of the signatures and the bound β in the underlying SIS assumption; this appears to be inherent to the style of the security reduction. Note also that the public keys in all of these schemes are still rather large at $\tilde{O}(n^3)$ bits (or $\tilde{O}(n^2)$ bits using the ring analogue of SIS), so they are still mainly of theoretical interest. Improving the key sizes of standard-model signatures is an important open problem.

Chosen ciphertext-secure encryption. We give a new construction of CCA-secure public-key encryption (in the standard model) from the learning with errors (LWE) problem with error rate $\alpha = 1/\text{poly}(n)$, where larger α corresponds to a harder concrete problem. Existing schemes exhibit various incomparable tradeoffs between key size and error rate. The first such scheme is due to [PW08]: it has public keys of size $\tilde{O}(n^2)$ bits (with somewhat large hidden factors) and relies on a quite small LWE error rate of $\alpha = \tilde{O}(1/n^4)$. The next scheme, from [Pei09b], has larger public keys of $\tilde{O}(n^3)$ bits, but uses a better error rate of $\alpha = \tilde{O}(1/n)$. Finally, using the generic conversion from selectively secure ID-based encryption to CCA-secure encryption [BCHK07], one can obtain from [ABB10a] a scheme having key size $\tilde{O}(n^2)$ bits and using error rate $\alpha = \tilde{O}(1/n^2)$. (Here decryption is randomized, since the IBE key-derivation algorithm is.) In particular, the public key of the scheme from [ABB10b] consists of 3 matrices in $\mathbb{Z}_q^{n \times m}$ where m is large enough to embed a (strong) trapdoor, plus essentially one vector in \mathbb{Z}_q^n per message bit.

We give a CCA-secure system that enjoys the best of all prior constructions, which has $\tilde{O}(n^2)$ -bit public keys, uses error rate $\alpha = \tilde{O}(1/n)$ (both with small hidden factors), and has deterministic decryption. To achieve this, we need to go beyond just plugging our improved trapdoor generator as a black box into prior constructions. Our scheme relies on the particular structure of the trapdoor instances; in effect, we directly construct a “tag-based adaptive trapdoor function” [KMO10]. The public key consists of only 1 matrix with an embedded (strong) trapdoor, rather than 3 as in the most compact scheme to date [ABB10a]; moreover, we can encrypt up to $n \log q$ message bits per ciphertext without needing any additional public key material. Combining these design changes with the improved dimension of our trapdoor generator, we obtain more than a 7.5-fold improvement in the public key size as compared with [ABB10a]. (This figure does not account for removing the extra public key material for the message bits, nor the other parameter improvements implied by our weaker concrete LWE assumption, which would shrink the keys even further.)

(Hierarchical) identity-based encryption. Just as with signatures, our constructions plug directly into the random-oracle IBE of [GPV08]. In the standard-model depth- d hierarchical IBEs of [CHKP10, ABB10a], our techniques can shrink the public parameters by an additional factor of about $\frac{2+4d}{1+d} \in [3, 4]$, relative to just plugging our improved trapdoor generator as a “black box” into the schemes. This is because for each level of the hierarchy, the public parameters only need to contain one matrix of the same dimension as \mathbf{G} (i.e., about $n \lg q$), rather than two full trapdoor matrices (of dimension about $2n \lg q$ each).³ Because the adaptation is straightforward given the tools developed in this work, we omit the details.

³We note that in [Pei09a] (an earlier version of [CHKP10]) the schemes are defined in a similar way using lower-dimensional extensions, rather than full trapdoor matrices at each level.

1.4 Other Related Work

Concrete parameter settings for a variety “strong” trapdoor applications are given in [RS10]. Those parameters are derived using the previous suboptimal generator of [AP09], and using the methods from this work would yield substantial improvements. The recent work of [LP11] also gives improved key sizes and concrete security for LWE-based cryptosystems; however, that work deals only with IND-CPA-secure encryption, and not at all with strong trapdoors or the further applications they enable (CCA security, digital signatures, (H)IBE, etc.).

2 Preliminaries

We denote the real numbers by \mathbb{R} and the integers by \mathbb{Z} . For a nonnegative integer k , we let $[k] = \{1, \dots, k\}$. Vectors are denoted by lower-case bold letters (e.g., \mathbf{x}) and are always in column form (\mathbf{x}^t is a row vector). We denote matrices by upper-case bold letters, and treat a matrix \mathbf{X} interchangeably with its ordered set $\{\mathbf{x}_1, \mathbf{x}_2, \dots\}$ of column vectors. For convenience, we sometimes use a scalar s to refer to the scaled identity matrix $s\mathbf{I}$, where the dimension will be clear from context.

The statistical distance between two distributions X, Y over a finite or countable domain D is $\Delta(X, Y) = \frac{1}{2} \sum_{w \in D} |X(w) - Y(w)|$. Statistical distance is a metric, and in particular obeys the triangle inequality. We say that a distribution over D is ϵ -uniform if its statistical distance from the uniform distribution is at most ϵ .

Throughout the paper, we use a “randomized-rounding parameter” r that we let be a fixed function $r(n) = \omega(\sqrt{\log n})$ growing asymptotically faster than $\sqrt{\log n}$. By “fixed function” we mean that $r = \omega(\sqrt{\log n})$ always refers to the very same function, and no other factors will be absorbed into the $\omega(\cdot)$ notation. This allows us to keep track of the precise multiplicative constants introduced by our constructions. Concretely, we take $r \approx \sqrt{\ln(2/\epsilon)/\pi}$ where ϵ is a desired bound on the statistical error introduced by each randomized-rounding operation for \mathbb{Z} , because the error is bounded by $\approx 2 \exp(-\pi r^2)$ according to Lemma 2.3 below. For example, for $\epsilon = 2^{-54}$ we have $r \leq 3.5$, and for $\epsilon = 2^{-71}$ we have $r \leq 4$.

2.1 Linear Algebra

A unimodular matrix $\mathbf{U} \in \mathbb{Z}^{m \times m}$ is one for which $|\det(\mathbf{U})| = 1$; in particular, $\mathbf{U}^{-1} \in \mathbb{Z}^{m \times m}$ as well. The *Gram-Schmidt orthogonalization* of an ordered set of vectors $\mathbf{V} = \{\mathbf{v}_1, \dots, \mathbf{v}_k\} \in \mathbb{R}^n$, is $\tilde{\mathbf{V}} = \{\tilde{\mathbf{v}}_1, \dots, \tilde{\mathbf{v}}_k\}$ where $\tilde{\mathbf{v}}_i$ is the component of \mathbf{v}_i orthogonal to $\text{span}(\mathbf{v}_1, \dots, \mathbf{v}_{i-1})$ for all $i = 1, \dots, k$. (In some cases we orthogonalize the vectors in a different order.) In matrix form, $\mathbf{V} = \mathbf{Q}\mathbf{D}\mathbf{U}$ for some orthogonal $\mathbf{Q} \in \mathbb{R}^{n \times k}$, diagonal $\mathbf{D} \in \mathbb{R}^{k \times k}$ with nonnegative entries, and upper unitriangular $\mathbf{U} \in \mathbb{R}^{k \times k}$ (i.e., \mathbf{U} is upper triangular with 1s on the diagonal). The decomposition is unique when the \mathbf{v}_i are linearly independent, and we always have $\|\tilde{\mathbf{v}}_i\| = d_{i,i}$, the i th diagonal entry of \mathbf{D} .

For any basis $\mathbf{V} = \{\mathbf{v}_1, \dots, \mathbf{v}_n\}$ of \mathbb{R}^n , its origin-centered parallelepiped is defined as $\mathcal{P}_{1/2}(\mathbf{V}) = \mathbf{V} \cdot [-\frac{1}{2}, \frac{1}{2}]^n$. Its dual basis is defined as $\mathbf{V}^* = \mathbf{V}^{-t} = (\mathbf{V}^{-1})^t$. If we orthogonalize \mathbf{V} and \mathbf{V}^* in forward and reverse order, respectively, then we have $\tilde{\mathbf{v}}_i^* = \tilde{\mathbf{v}}_i / \|\tilde{\mathbf{v}}_i\|^2$ for all i . In particular, $\|\tilde{\mathbf{v}}_i^*\| = 1 / \|\tilde{\mathbf{v}}_i\|$.

For any square real matrix \mathbf{X} , the (*Moore-Penrose*) *pseudoinverse*, denoted \mathbf{X}^+ , is the unique matrix satisfying $(\mathbf{X}\mathbf{X}^+)\mathbf{X} = \mathbf{X}$, $\mathbf{X}^+(\mathbf{X}\mathbf{X}^+) = \mathbf{X}^+$, and such that both $\mathbf{X}\mathbf{X}^+$ and $\mathbf{X}^+\mathbf{X}$ are symmetric. We always have $\text{span}(\mathbf{X}) = \text{span}(\mathbf{X}^+)$, and when \mathbf{X} is invertible, we have $\mathbf{X}^+ = \mathbf{X}^{-1}$.

A symmetric matrix $\Sigma \in \mathbb{R}^{n \times n}$ is *positive definite* (respectively, positive *semidefinite*), written $\Sigma > \mathbf{0}$ (resp., $\Sigma \geq \mathbf{0}$), if $\mathbf{x}^t \Sigma \mathbf{x} > 0$ (resp., $\mathbf{x}^t \Sigma \mathbf{x} \geq 0$) for all nonzero $\mathbf{x} \in \mathbb{R}^n$. We have $\Sigma > \mathbf{0}$ if and only if Σ is invertible and $\Sigma^{-1} > \mathbf{0}$, and $\Sigma \geq \mathbf{0}$ if and only if $\Sigma^+ \geq \mathbf{0}$. Positive (semi)definiteness defines a partial

ordering on symmetric matrices: we say that $\Sigma_1 > \Sigma_2$ if $(\Sigma_1 - \Sigma_2) > \mathbf{0}$, and similarly for $\Sigma_1 \geq \Sigma_2$. We have $\Sigma_1 \geq \Sigma_2 \geq \mathbf{0}$ if and only if $\Sigma_2^+ \geq \Sigma_1^+ \geq \mathbf{0}$, and likewise for the analogous strict inequalities.

For any matrix \mathbf{B} , the symmetric matrix $\Sigma = \mathbf{B}\mathbf{B}^t$ is positive semidefinite, because

$$\mathbf{x}^t \Sigma \mathbf{x} = \langle \mathbf{B}^t \mathbf{x}, \mathbf{B}^t \mathbf{x} \rangle = \|\mathbf{B}^t \mathbf{x}\|^2 \geq 0$$

for any nonzero $\mathbf{x} \in \mathbb{R}^n$, where the inequality is always strict if and only if \mathbf{B} is nonsingular. We say that \mathbf{B} is a *square root* of $\Sigma > \mathbf{0}$, written $\mathbf{B} = \sqrt{\Sigma}$, if $\mathbf{B}\mathbf{B}^t = \Sigma$. Every $\Sigma \geq \mathbf{0}$ has a square root, which can be computed efficiently, e.g., via the Cholesky decomposition.

For any matrix $\mathbf{B} \in \mathbb{R}^{n \times k}$, there exists a *singular value decomposition* $\mathbf{B} = \mathbf{Q}\mathbf{D}\mathbf{P}^t$, where $\mathbf{Q} \in \mathbb{R}^{n \times n}$, $\mathbf{P} \in \mathbb{R}^{k \times k}$ are orthogonal matrices, and $\mathbf{D} \in \mathbb{R}^{n \times k}$ is a diagonal matrix with nonnegative entries $s_i \geq 0$ on the diagonal, in non-increasing order. The s_i are called the *singular values* of \mathbf{B} . Under this convention, \mathbf{D} is uniquely determined (though \mathbf{Q}, \mathbf{P} may not be), and $s_1(\mathbf{B}) = \max_{\mathbf{u}} \|\mathbf{B}\mathbf{u}\| = \max_{\mathbf{u}} \|\mathbf{B}^t \mathbf{u}\| \geq \|\mathbf{B}\|, \|\mathbf{B}^t\|$, where the maxima are taken over all unit vectors $\mathbf{u} \in \mathbb{R}^k$.

2.2 Lattices and Hard Problems

Generally defined, an m -dimensional *lattice* Λ is a discrete additive subgroup of \mathbb{R}^m . For some $k \leq m$, called the *rank* of the lattice, Λ is generated as the set of all \mathbb{Z} -linear combinations of some k linearly independent *basis* vectors $\mathbf{B} = \{\mathbf{b}_1, \dots, \mathbf{b}_k\}$, i.e., $\Lambda = \{\mathbf{B}\mathbf{z} : \mathbf{z} \in \mathbb{Z}^k\}$. In this work, we are mostly concerned with full-rank integer lattices, i.e., $\Lambda \subseteq \mathbb{Z}^m$ with $k = m$. (We work with non-full-rank lattices only in the analysis of our Gaussian sampling algorithm in Section 5.4.) The dual lattice Λ^* is the set of all $\mathbf{v} \in \text{span}(\Lambda)$ such that $\langle \mathbf{v}, \mathbf{x} \rangle \in \mathbb{Z}$ for every $\mathbf{x} \in \Lambda$. If \mathbf{B} is a basis of Λ , then $\mathbf{B}^* = \mathbf{B}(\mathbf{B}^t \mathbf{B})^{-1}$ is a basis of Λ^* . Note that when Λ is full-rank, \mathbf{B} is invertible and hence $\mathbf{B}^* = \mathbf{B}^{-t}$.

Many cryptographic applications use a particular family of so-called q -ary integer lattices, which contain $q\mathbb{Z}^m$ as a sublattice for some (typically small) integer q . For positive integers n and q , let $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ be arbitrary and define the following full-rank m -dimensional q -ary lattices:

$$\begin{aligned} \Lambda^\perp(\mathbf{A}) &= \{\mathbf{z} \in \mathbb{Z}^m : \mathbf{A}\mathbf{z} = \mathbf{0} \bmod q\} \\ \Lambda(\mathbf{A}^t) &= \{\mathbf{z} \in \mathbb{Z}^m : \exists \mathbf{s} \in \mathbb{Z}_q^n \text{ s.t. } \mathbf{z} = \mathbf{A}^t \mathbf{s} \bmod q\}. \end{aligned}$$

It is easy to check that $\Lambda^\perp(\mathbf{A})$ and $\Lambda(\mathbf{A}^t)$ are dual lattices, up to a q scaling factor: $q \cdot \Lambda^\perp(\mathbf{A})^* = \Lambda(\mathbf{A}^t)$, and vice-versa. For this reason, it is sometimes more natural to consider the non-integral, “1-ary” lattice $\frac{1}{q}\Lambda(\mathbf{A}^t) = \Lambda^\perp(\mathbf{A})^* \supseteq \mathbb{Z}^m$. For any $\mathbf{u} \in \mathbb{Z}_q^n$ admitting an integral solution to $\mathbf{A}\mathbf{x} = \mathbf{u} \bmod q$, define the coset (or “shifted” lattice)

$$\Lambda_{\mathbf{u}}^\perp(\mathbf{A}) = \{\mathbf{z} \in \mathbb{Z}^m : \mathbf{A}\mathbf{z} = \mathbf{u} \bmod q\} = \Lambda^\perp(\mathbf{A}) + \mathbf{x}.$$

Here we recall some basic facts about these q -ary lattices.

Lemma 2.1. *Let $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ be arbitrary and let $\mathbf{S} \in \mathbb{Z}^{m \times m}$ be any basis of $\Lambda^\perp(\mathbf{A})$.*

1. *For any unimodular $\mathbf{T} \in \mathbb{Z}^{m \times m}$, we have $\mathbf{T} \cdot \Lambda^\perp(\mathbf{A}) = \Lambda^\perp(\mathbf{A} \cdot \mathbf{T}^{-1})$, with $\mathbf{T} \cdot \mathbf{S}$ as a basis.*
2. *[ABB10a, implicit] For any invertible $\mathbf{H} \in \mathbb{Z}_q^{n \times n}$, we have $\Lambda^\perp(\mathbf{H} \cdot \mathbf{A}) = \Lambda^\perp(\mathbf{A})$.*
3. *[CHKP10, Lemma 3.2] Suppose that the columns of \mathbf{A} generate all of \mathbb{Z}_q^n , let $\mathbf{A}' \in \mathbb{Z}_q^{n \times m'}$ be arbitrary, and let $\mathbf{W} \in \mathbb{Z}^{m \times m'}$ be an arbitrary solution to $\mathbf{A}\mathbf{W} = -\mathbf{A}' \bmod q$. Then $\mathbf{S}' = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{W} & \mathbf{S} \end{bmatrix}$ is a basis of $\Lambda^\perp([\mathbf{A}' \mid \mathbf{A}])$, and when orthogonalized in appropriate order, $\tilde{\mathbf{S}}' = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \tilde{\mathbf{S}} \end{bmatrix}$. In particular, $\|\tilde{\mathbf{S}}'\| = \|\tilde{\mathbf{S}}\|$.*

Cryptographic problems. For $\beta > 0$, the *short integer solution* problem $\text{SIS}_{q,\beta}$ is an average-case version of the approximate shortest vector problem on $\Lambda^\perp(\mathbf{A})$. The problem is: given uniformly random $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ for any desired $m = \text{poly}(n)$, find a relatively short nonzero $\mathbf{z} \in \Lambda^\perp(\mathbf{A})$, i.e., output a nonzero $\mathbf{z} \in \mathbb{Z}^m$ such that $\mathbf{A}\mathbf{z} = \mathbf{0} \bmod q$ and $\|\mathbf{z}\| \leq \beta$. When $q \geq \beta\sqrt{n} \cdot \omega(\sqrt{\log n})$, solving this problem (with any non-negligible probability over the random choice of \mathbf{A}) is at least as hard as (probabilistically) approximating the Shortest Independent Vectors Problem (SIVP, a classic problem in the computational study of point lattices [MG02]) on n -dimensional lattices to within $\tilde{O}(\beta\sqrt{n})$ factors in the *worst case* [Ajt96, MR04, GPV08].

For $\alpha > 0$, the *learning with errors* problem $\text{LWE}_{q,\alpha}$ may be seen an average-case version of the bounded-distance decoding problem on the dual lattice $\frac{1}{q}\Lambda(\mathbf{A}^t)$. Let $\mathbb{T} = \mathbb{R}/\mathbb{Z}$, the additive group of reals modulo 1, and let D_α denote the Gaussian probability distribution over \mathbb{R} with parameter α (see Section 2.3 below). For any fixed $\mathbf{s} \in \mathbb{Z}_q^n$, define $A_{\mathbf{s},\alpha}$ to be the distribution over $\mathbb{Z}_q^n \times \mathbb{T}$ obtained by choosing $\mathbf{a} \leftarrow \mathbb{Z}_q^n$ uniformly at random, choosing $e \leftarrow D_\alpha$, and outputting $(\mathbf{a}, b = \langle \mathbf{a}, \mathbf{s} \rangle / q + e \bmod 1)$. The search- $\text{LWE}_{q,\alpha}$ problem is: given any desired number $m = \text{poly}(n)$ of independent samples from $A_{\mathbf{s},\alpha}$ for some arbitrary \mathbf{s} , find \mathbf{s} . The decision- $\text{LWE}_{q,\alpha}$ problem is to distinguish, with non-negligible advantage, between samples from $A_{\mathbf{s},\alpha}$ for uniformly random $\mathbf{s} \in \mathbb{Z}_q^n$, and uniformly random samples from $\mathbb{Z}_q^n \times \mathbb{T}$. There are a variety of (incomparable) search/decision reductions for LWE under certain conditions on the parameters (e.g., [Reg05, Pei09b, ACPS09]); in Section 3 we give a reduction that essentially subsumes them all. When $q \geq 2\sqrt{n}/\alpha$, solving search- $\text{LWE}_{q,\alpha}$ is at least as hard as *quantumly* approximating SIVP on n -dimensional lattices to within $\tilde{O}(n/\alpha)$ factors in the worst case [Reg05]. For a restricted range of parameters (e.g., when q is exponentially large) a *classical* (non-quantum) reduction is also known [Pei09b], but only from a potentially easier class of problems like the decisional Shortest Vector Problem (GapSVP) and the Bounded Distance Decoding Problem (BDD) (see [LM09]).

Note that the m samples (\mathbf{a}_i, b_i) and underlying error terms e_i from $A_{\mathbf{s},\alpha}$ may be grouped into a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ and vectors $\mathbf{b} \in \mathbb{T}^m$, $\mathbf{e} \in \mathbb{R}^m$ in the natural way, so that $\mathbf{b} = (\mathbf{A}^t \mathbf{s})/q + \mathbf{e} \bmod 1$. In this way, \mathbf{b} may be seen as an element of $\Lambda^\perp(\mathbf{A})^* = \frac{1}{q}\Lambda(\mathbf{A}^t)$, perturbed by Gaussian error. By scaling \mathbf{b} and discretizing its entries using a form of randomized rounding (see [Pei10]), we can convert it into $\mathbf{b}' = \mathbf{A}^t \mathbf{s} + \mathbf{e}' \bmod q$ where $\mathbf{e}' \in \mathbb{Z}^m$ has *discrete* Gaussian distribution with parameter (say) $\sqrt{2}\alpha q$.

2.3 Gaussians and Lattices

The n -dimensional Gaussian function $\rho : \mathbb{R}^n \rightarrow (0, 1]$ is defined as

$$\rho(\mathbf{x}) \triangleq \exp(-\pi \cdot \|\mathbf{x}\|^2) = \exp(-\pi \cdot \langle \mathbf{x}, \mathbf{x} \rangle).$$

Applying a linear transformation given by a (not necessarily square) matrix \mathbf{B} with linearly independent columns yields the (possibly degenerate) Gaussian function

$$\rho_{\mathbf{B}}(\mathbf{x}) \triangleq \begin{cases} \rho(\mathbf{B}^+ \mathbf{x}) = \exp(-\pi \cdot \mathbf{x}^t \Sigma^+ \mathbf{x}) & \text{if } \mathbf{x} \in \text{span}(\mathbf{B}) = \text{span}(\Sigma) \\ 0 & \text{otherwise} \end{cases}$$

where $\Sigma = \mathbf{B}\mathbf{B}^t \geq \mathbf{0}$. Because $\rho_{\mathbf{B}}$ is distinguished only up to Σ , we usually refer to it as $\rho_{\sqrt{\Sigma}}$.

Normalizing $\rho_{\sqrt{\Sigma}}$ by its total measure over $\text{span}(\Sigma)$, we obtain the probability distribution function of the (continuous) *Gaussian distribution* $D_{\sqrt{\Sigma}}$. By linearity of expectation, this distribution has *covariance* $\mathbb{E}_{\mathbf{x} \leftarrow D_{\sqrt{\Sigma}}}[\mathbf{x} \cdot \mathbf{x}^t] = \frac{\Sigma}{2\pi}$. (The $\frac{1}{2\pi}$ factor is the variance of the Gaussian D_1 , due to our choice of normalization.) For convenience, we implicitly ignore the $\frac{1}{2\pi}$ factor, and refer to Σ as the covariance matrix of $D_{\sqrt{\Sigma}}$.

Let $\Lambda \subset \mathbb{R}^n$ be a lattice, let $\mathbf{c} \in \mathbb{R}^n$, and let $\Sigma \geq \mathbf{0}$ be a positive semidefinite matrix such that $(\Lambda + \mathbf{c}) \cap \text{span}(\Sigma)$ is nonempty. The *discrete Gaussian distribution* $D_{\Lambda+\mathbf{c}, \sqrt{\Sigma}}$ is simply the Gaussian distribution $D_{\sqrt{\Sigma}}$ restricted to have support $\Lambda + \mathbf{c}$. That is, for all $\mathbf{x} \in \Lambda + \mathbf{c}$,

$$D_{\Lambda+\mathbf{c}, \sqrt{\Sigma}}(\mathbf{x}) = \frac{\rho_{\sqrt{\Sigma}}(\mathbf{x})}{\rho_{\sqrt{\Sigma}}(\Lambda + \mathbf{c})} \propto \rho_{\sqrt{\Sigma}}(\mathbf{x}).$$

We recall the definition of the *smoothing parameter* from [MR04], generalized to non-spherical (and potentially degenerate) Gaussians. It is easy to see that the definition is consistent with the partial ordering of positive semidefinite matrices, i.e., if $\Sigma_1 \geq \Sigma_2 \geq \eta_\epsilon(\Lambda)$, then $\Sigma_1 \geq \eta_\epsilon(\Lambda)$.

Definition 2.2. Let $\Sigma \geq \mathbf{0}$ and $\Lambda \subset \text{span}(\Sigma)$ be a lattice. We say that $\sqrt{\Sigma} \geq \eta_\epsilon(\Lambda)$ if $\rho_{\sqrt{\Sigma}}(\Lambda^*) \leq 1 + \epsilon$.

The following is a bound on the smoothing parameter in terms of any orthogonalized basis. Note that for practical choices like $n \leq 2^{14}$ and $\epsilon \geq 2^{-80}$, the multiplicative factor attached to $\|\tilde{\mathbf{B}}\|$ is bounded by 4.6.

Lemma 2.3 ([GPV08, Theorem 3.1]). *Let $\Lambda \subset \mathbb{R}^n$ be a lattice with basis \mathbf{B} , and let $\epsilon > 0$. We have*

$$\eta_\epsilon(\Lambda) \leq \|\tilde{\mathbf{B}}\| \cdot \sqrt{\ln(2n(1 + 1/\epsilon))/\pi}.$$

In particular, for any $\omega(\sqrt{\log n})$ function, there is a negligible $\epsilon(n)$ for which $\eta_\epsilon(\Lambda) \leq \|\tilde{\mathbf{B}}\| \cdot \omega(\sqrt{\log n})$.

For appropriate parameters, the smoothing parameter of a random lattice $\Lambda^\perp(\mathbf{A})$ is small, with very high probability. The following bound is a refinement and strengthening of one from [GPV08], which allows for a more precise analysis of the parameters and statistical errors involved in our constructions.

Lemma 2.4. *Let $n, m, q \geq 2$ be positive integers. For $\mathbf{s} \in \mathbb{Z}_q^n$, let the subgroup $\mathbb{G}_{\mathbf{s}} = \{\langle \mathbf{a}, \mathbf{s} \rangle : \mathbf{a} \in \mathbb{Z}_q^n\} \subseteq \mathbb{Z}_q$, and let $g_{\mathbf{s}} = |\mathbb{G}_{\mathbf{s}}| = q / \gcd(s_1, \dots, s_n, q)$. Let $\epsilon > 0$, $\eta \geq \eta_\epsilon(\mathbb{Z}^m)$, and $s > \eta$ be reals. Then for uniformly random $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$,*

$$\mathbb{E}_{\mathbf{A}} \left[\rho_{1/s}(\Lambda^\perp(\mathbf{A})^*) \right] \leq (1 + \epsilon) \sum_{\mathbf{s} \in \mathbb{Z}_q^n} \max\{1/g_{\mathbf{s}}, \eta/s\}^m. \quad (2.1)$$

In particular, if $q = p^e$ is a power of a prime p , and

$$m \geq \max \left\{ n + \frac{\log(3 + 2/\epsilon)}{\log p}, \frac{n \log q + \log(2 + 2/\epsilon)}{\log(s/\eta)} \right\}, \quad (2.2)$$

then $\mathbb{E}_{\mathbf{A}} [\rho_{1/s}(\Lambda^\perp(\mathbf{A})^)] \leq 1 + 2\epsilon$, and so by Markov's inequality, $s \geq \eta_{2\epsilon/\delta}(\Lambda^\perp(\mathbf{A}))$ except with probability at most δ .*

Proof. We will use the fact (which follows from the Poisson summation formula; see [MR04, Lemma 2.8]) that $\rho_t(\Lambda) \leq \rho_r(\Lambda) \leq (r/t)^m \cdot \rho_t(\Lambda)$ for any rank- m lattice Λ and $r \geq t > 0$.

For any $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, one can check that $\Lambda^\perp(\mathbf{A})^* = \mathbb{Z}^m + \{\mathbf{A}^t \mathbf{s} / q : \mathbf{s} \in \mathbb{Z}_q^n\}$. Note that $\mathbf{A}^t \mathbf{s}$ is uniformly

random over \mathbb{G}_s^m , for uniformly random \mathbf{A} . Then we have

$$\begin{aligned}
\mathbb{E}_{\mathbf{A}} \left[\rho_{1/s}(\Lambda^\perp(\mathbf{A})^*) \right] &\leq \sum_{\mathbf{s} \in \mathbb{Z}_q^n} \mathbb{E}_{\mathbf{A}} \left[\rho_{1/s}(\mathbb{Z}^m + \mathbf{A}^t \mathbf{s}/q) \right] && (\text{lin. of } \mathbb{E}) \\
&= \sum_{\mathbf{s} \in \mathbb{Z}_q^n} g_{\mathbf{s}}^{-m} \cdot \rho_{1/s}(g_{\mathbf{s}}^{-1} \cdot \mathbb{Z}^m) && (\text{avg. over } \mathbf{A}) \\
&\leq \sum_{\mathbf{s} \in \mathbb{Z}_q^n} g_{\mathbf{s}}^{-m} \cdot \max\{1, g_{\mathbf{s}} \eta/s\}^m \cdot \rho_{1/\eta}(\mathbb{Z}^m), && (\text{above fact}) \\
&\leq (1 + \epsilon) \sum_{\mathbf{s} \in \mathbb{Z}_q^n} \max\{1/g_{\mathbf{s}}, \eta/s\}^m, && (\eta \geq \eta_\epsilon(\mathbb{Z}^m)).
\end{aligned}$$

To prove the second part of the claim, observe that $g_{\mathbf{s}} = p^i$ for some $i \geq 0$, and that there are at most g^n values of \mathbf{s} for which $g_{\mathbf{s}} = g$, because each entry of \mathbf{s} must be in \mathbb{G}_s . Therefore,

$$\sum_{\mathbf{s} \in \mathbb{Z}_q^n} 1/g_{\mathbf{s}}^m \leq \sum_{i \geq 0} p^{i(n-m)} = \frac{1}{1 - p^{n-m}} \leq 1 + \frac{\epsilon}{2(1 + \epsilon)}.$$

(More generally, for arbitrary q we have $\sum_{\mathbf{s}} 1/g_{\mathbf{s}}^m \leq \zeta(m - n)$, where $\zeta(\cdot)$ is the Riemann zeta function.) Similarly, $\sum_{\mathbf{s}} (\eta/s)^m = q^n (s/\eta)^{-m} \leq \frac{\epsilon}{2(1+\epsilon)}$, and the claim follows. \square

We need a number of standard facts about discrete Gaussians.

Lemma 2.5 ([MR04, Lemmas 2.9 and 4.1]). *Let $\Lambda \subset \mathbb{R}^n$ be a lattice. For any $\Sigma \geq \mathbf{0}$ and $\mathbf{c} \in \mathbb{R}^n$, we have $\rho_{\sqrt{\Sigma}}(\Lambda + \mathbf{c}) \leq \rho_{\sqrt{\Sigma}}(\Lambda)$. Moreover, if $\sqrt{\Sigma} \geq \eta_\epsilon(\Lambda)$ for some $\epsilon > 0$ and $\mathbf{c} \in \text{span}(\Lambda)$, then $\rho_{\sqrt{\Sigma}}(\Lambda + \mathbf{c}) \geq \frac{1-\epsilon}{1+\epsilon} \cdot \rho_{\sqrt{\Sigma}}(\Lambda)$.*

Combining the above lemma with a bound of Banaszczyk [Ban93], we have the following tail bound on discrete Gaussians.

Lemma 2.6 ([Ban93, Lemma 1.5]). *Let $\Lambda \subset \mathbb{R}^n$ be a lattice and $r \geq \eta_\epsilon(\Lambda)$ for some $\epsilon \in (0, 1)$. For any $\mathbf{c} \in \text{span}(\Lambda)$, we have*

$$\Pr \left[\|D_{\Lambda+\mathbf{c},r}\| \geq r\sqrt{n} \right] \leq 2^{-n} \cdot \frac{1+\epsilon}{1-\epsilon}.$$

Moreover, if $\mathbf{c} = \mathbf{0}$ then the bound holds for any $r > 0$, with $\epsilon = 0$.

The next lemma bounds the predictability (i.e., probability of the most likely outcome or equivalently, min-entropy) of a discrete Gaussian.

Lemma 2.7 ([PR06, Lemma 2.11]). *Let $\Lambda \subset \mathbb{R}^n$ be a lattice and $r \geq 2\eta_\epsilon(\Lambda)$ for some $\epsilon \in (0, 1)$. For any $\mathbf{c} \in \mathbb{R}^n$ and any $\mathbf{y} \in \Lambda + \mathbf{c}$, we have $\Pr[D_{\Lambda+\mathbf{c},r} = \mathbf{y}] \leq 2^{-n} \cdot \frac{1+\epsilon}{1-\epsilon}$.*

2.4 Subgaussian Distributions and Random Matrices

For $\delta \geq 0$, we say that a random variable X (or its distribution) over \mathbb{R} is δ -subgaussian with parameter $s > 0$ if for all $t \in \mathbb{R}$, the (scaled) moment-generating function satisfies

$$\mathbb{E}[\exp(2\pi t X)] \leq \exp(\delta) \cdot \exp(\pi s^2 t^2).$$

Notice that the $\exp(\pi s^2 t^2)$ term on the right is precisely the (scaled) moment-generating function of the Gaussian distribution D_s . So, our definition differs from the usual definition of subgaussian only in the additional factor of $\exp(\delta)$; we need this relaxation when working with discrete Gaussians, usually taking $\delta = \ln(\frac{1+\epsilon}{1-\epsilon}) \approx 2\epsilon$ for the same small ϵ as in the smoothing parameter η_ϵ .

If X is δ -subgaussian, then its tails are dominated by a Gaussian of parameter s , i.e., $\Pr[|X| \geq t] \leq 2\exp(\delta)\exp(-\pi t^2/s^2)$ for all $t \geq 0$.⁴ This follows by Markov's inequality: by scaling X we can assume $s = 1$, and we have

$$\Pr[X \geq t] = \Pr[\exp(2\pi t X) \geq \exp(2\pi t^2)] \leq \exp(\delta) \exp(\pi t^2) / \exp(2\pi t^2) = \exp(\delta) \exp(-\pi t^2).$$

The claim follows by repeating the argument with $-X$, and the union bound. Using the Taylor series expansion of $\exp(2\pi t X)$, it can be shown that any B -bounded symmetric random variable X (i.e., $|X| \leq B$ always) is 0-subgaussian with parameter $B\sqrt{2\pi}$.

More generally, we say that a random vector \mathbf{x} or its distribution (respectively, a random matrix \mathbf{X}) is δ -subgaussian (of parameter s) if all its one-dimensional marginals $\langle \mathbf{u}, \mathbf{v} \rangle$ (respectively, $\mathbf{u}^t \mathbf{X} \mathbf{v}$) for unit vectors \mathbf{u}, \mathbf{v} are δ -subgaussian (of parameter s). It follows immediately from the definition that the concatenation of independent δ_i -subgaussian vectors with common parameter s , interpreted as either a vector or matrix, is $(\sum \delta_i)$ -subgaussian with parameter s .

Lemma 2.8. *Let $\Lambda \subset \mathbb{R}^n$ be a lattice and $s \geq \eta_\epsilon(\Lambda)$ for some $0 < \epsilon < 1$. For any $\mathbf{c} \in \text{span}(\Lambda)$, $D_{\Lambda+\mathbf{c},s}$ is $\ln(\frac{1+\epsilon}{1-\epsilon})$ -subgaussian with parameter s . Moreover, it is 0-subgaussian for any $s > 0$ when $\mathbf{c} = \mathbf{0}$.*

Proof. By scaling Λ we can assume that $s = 1$. Let \mathbf{x} have distribution $D_{\Lambda+\mathbf{c}}$, and let $\mathbf{u} \in \mathbb{R}^n$ be any unit vector. We bound the scaled moment-generating function of the marginal $\langle \mathbf{x}, \mathbf{u} \rangle$ for any $t \in \mathbb{R}$:

$$\begin{aligned} \rho(\Lambda + \mathbf{c}) \cdot \mathbb{E}[\exp(2\pi \langle \mathbf{x}, t\mathbf{u} \rangle)] &= \sum_{\mathbf{x} \in \Lambda + \mathbf{c}} \exp(-\pi(\langle \mathbf{x}, \mathbf{x} \rangle - 2\langle \mathbf{x}, t\mathbf{u} \rangle)) \\ &= \exp(\pi t^2) \cdot \sum_{\mathbf{x} \in \Lambda + \mathbf{c}} \exp(-\pi \langle \mathbf{x} - t\mathbf{u}, \mathbf{x} - t\mathbf{u} \rangle) \\ &= \exp(\pi t^2) \cdot \rho(\Lambda + \mathbf{c} - t\mathbf{u}). \end{aligned}$$

Both claims then follow by Lemma 2.5. □

Here we recall a standard result from the non-asymptotic theory of random matrices; for further details, see [Ver11]. (The proof for δ -subgaussian distributions is a trivial adaptation of the 0-subgaussian case.)

Lemma 2.9. *Let $\mathbf{X} \in \mathbb{R}^{n \times m}$ be a δ -subgaussian random matrix with parameter s . There exists a universal constant $C > 0$ such that for any $t \geq 0$, we have $s_1(\mathbf{X}) \leq C \cdot s \cdot (\sqrt{m} + \sqrt{n} + t)$ except with probability at most $2\exp(\delta)\exp(-\pi t^2)$.*

Empirically, for discrete Gaussians the universal constant C in the above lemma is very close to $1/\sqrt{2\pi}$. In fact, it has been proved that $C \leq 1/\sqrt{2\pi}$ for matrices with independent identically distributed *continuous* Gaussian entries.

⁴The converse also holds (up to a small constant factor in the parameter s) when $\mathbb{E}[X] = 0$, but this will frequently not quite be the case in our applications, which is why we define subgaussian in terms of the moment-generating function.

3 Search to Decision Reduction

Here we give a new search-to-decision reduction for LWE that essentially subsumes all of the (incomparable) prior ones given in [BFKL93, Reg05, Pei09b, ACPS09].⁵ Most notably, it handles moduli q that were not covered before, specifically, those like $q = 2^k$ that are divisible by powers of very small primes. The only known reduction that ours does not subsume is a different style of *sample-preserving* reduction recently given in [MM11], which works for a more limited class of moduli and error distributions; extending that reduction to the full range of parameters considered here is an interesting open problem. In what follows, $\omega(\sqrt{\log n})$ denotes some fixed function that grows faster than $\sqrt{\log n}$, asymptotically.

Theorem 3.1. *Let q have prime factorization $q = p_1^{e_1} \cdots p_k^{e_k}$ for pairwise distinct $\text{poly}(n)$ -bounded primes p_i with each $e_i \geq 1$, and let $0 < \alpha \leq 1/\omega(\sqrt{\log n})$. Let ℓ be the number of prime factors $p_i < \omega(\sqrt{\log n})/\alpha$. There is a probabilistic polynomial-time reduction from solving search-LWE $_{q,\alpha}$ (in the worst case, with overwhelming probability) to solving decision-LWE $_{q,\alpha'}$ (on the average, with non-negligible advantage) for any $\alpha' \geq \alpha$ such that $\alpha' \geq \omega(\sqrt{\log n})/p_i^{e_i}$ for every i , and $(\alpha')^\ell \geq \alpha \cdot \omega(\sqrt{\log n})^{1+\ell}$.*

For example, when every $p_i \geq \omega(\sqrt{\log n})/\alpha$ we have $\ell = 0$, and any $\alpha' \geq \alpha$ is acceptable. (This special case, with the additional constraint that every $e_i = 1$, is proved in [Pei09b].) As a qualitatively new example, when $q = p^e$ is a prime power for some (possibly small) prime p , then it suffices to let $\alpha' \geq \alpha \cdot \omega(\sqrt{\log n})^2$. (A similar special case where $q = p^e$ for sufficiently large p and $\alpha' = \alpha \ll 1/p$ is proved in [ACPS09].)

Proof. We show how to recover each entry of \mathbf{s} modulo a large enough power of each p_i , given access to the distribution $A_{\mathbf{s},\alpha}$ for some $\mathbf{s} \in \mathbb{Z}_q^n$ and to an oracle \mathcal{O} solving DLWE $_{q,\alpha'}$. For the parameters in the theorem statement, we can then recover the remainder of \mathbf{s} in polynomial time by rounding and standard Gaussian elimination.

First, observe that we can transform $A_{\mathbf{s},\alpha}$ into $A_{\mathbf{s},\alpha'}$ simply by adding (modulo 1) an independent sample from $D_{\sqrt{\alpha'^2 - \alpha^2}}$ to the second component of each $(\mathbf{a}, b = \langle \mathbf{a}, \mathbf{s} \rangle / q + D_\alpha \bmod 1) \in \mathbb{Z}_q^n \times \mathbb{T}$ drawn from $A_{\mathbf{s},\alpha}$.

We now show how to recover each entry of \mathbf{s} modulo (powers of) any prime $p = p_i$ dividing q . Let $e = e_i$, and for $j = 0, 1, \dots, e$ define $A_{\mathbf{s},\alpha'}^j$ to be the distribution over $\mathbb{Z}_q^n \times \mathbb{T}$ obtained by drawing $(\mathbf{a}, b) \leftarrow A_{\mathbf{s},\alpha'}$ and outputting $(\mathbf{a}, b + r/p^j \bmod 1)$ for a fresh uniformly random $r \leftarrow \mathbb{Z}_q$. (Clearly, this distribution can be generated efficiently from $A_{\mathbf{s},\alpha'}$.) Note that when $\alpha' \geq \omega(\sqrt{\log n})/p^j \geq \eta_\epsilon((1/p^j)\mathbb{Z})$ for some $\epsilon = \text{negl}(n)$, $A_{\mathbf{s},\alpha'}^j$ is negligibly far from $U = U(\mathbb{Z}_q^n \times \mathbb{T})$, and this holds at least for $j = e$ by hypothesis. Therefore, by a hybrid argument there exists some minimal $j \in [e]$ for which \mathcal{O} has a non-negligible advantage in distinguishing between $A_{\mathbf{s},\alpha'}^{j-1}$ and $A_{\mathbf{s},\alpha'}^j$, over a random choice of \mathbf{s} and all other randomness in the experiment. (This j can be found efficiently by measuring the behavior of \mathcal{O} .) Note that when $p_i \geq \omega(\sqrt{\log n})/\alpha \geq \omega(\sqrt{\log n})/\alpha'$, the minimal j must be 1; otherwise it may be larger, but there are at most ℓ of these by hypothesis. Now by a standard random self-reduction and amplification techniques (e.g., [Reg05, Lemma 4.1]), we can in fact assume that \mathcal{O} accepts (respectively, rejects) with *overwhelming* probability given $A_{\mathbf{s},\alpha'}^{j-1}$ (resp., $A_{\mathbf{s},\alpha'}^j$), for any $\mathbf{s} \in \mathbb{Z}_q^n$.

Given access to $A_{\mathbf{s},\alpha'}^{j-1}$ and \mathcal{O} , we can test whether $s_1 = 0 \bmod p$ by invoking \mathcal{O} on samples from $A_{\mathbf{s},\alpha'}^{j-1}$ that have been transformed as follows (all of what follows is analogous for s_2, \dots, s_n): take each sample $(\mathbf{a}, b = \langle \mathbf{a}, \mathbf{s} \rangle / q + e + r/p^{j-1} \bmod 1) \leftarrow A_{\mathbf{s},\alpha'}^{j-1}$ to

$$(\mathbf{a}' = \mathbf{a} - r' \cdot (q/p^j) \cdot \mathbf{e}_1, \quad b' = b = \langle \mathbf{a}', \mathbf{s} \rangle / q + e + (pr + r's_1)/p^j \bmod 1) \quad (3.1)$$

⁵We say “essentially subsumes” because our reduction is not very meaningful when q is itself a very small prime, whereas those of [BFKL93, Reg05] are meaningful. This is only because our reduction deals with the *continuous* version of LWE. If we discretize the problem, then for very small prime q our reduction specializes to those of [BFKL93, Reg05].

for a fresh $r' \leftarrow \mathbb{Z}_q$ (where $\mathbf{e}_1 = (1, 0, \dots, 0) \in \mathbb{Z}_q^n$). Observe that if $s_1 = 0 \bmod p$, the transformed samples are also drawn from $A_{\mathbf{s}, \alpha'}^{j-1}$, otherwise they are drawn from $A_{\mathbf{s}, \alpha'}^j$ because $r's_1$ is uniformly random modulo p . Therefore, \mathcal{O} tells us which is the case.

Using the above test, we can efficiently recover $s_1 \bmod p$ by ‘shifting’ s_1 by each of $0, \dots, p-1 \bmod p$ using the standard transformation that maps $A_{\mathbf{s}, \alpha'}$ to $A_{\mathbf{s}+\mathbf{t}, \alpha'}$ for any desired $\mathbf{t} \in \mathbb{Z}_q^n$, by taking (\mathbf{a}, b) to $(\mathbf{a}, b + \langle \mathbf{a}, \mathbf{t} \rangle / q \bmod 1)$. (This enumeration step is where we use the fact that every p_i is $\text{poly}(n)$ -bounded.) Moreover, we can iteratively recover $s_1 \bmod p^2, \dots, p^{e-j+1}$ as follows: having recovered $s_1 \bmod p^i$, first ‘shift’ $A_{\mathbf{s}, \alpha'}$ to $A_{\mathbf{s}', \alpha'}$ where $s'_1 = 0 \bmod p^i$, then apply a similar procedure as above to recover $s'_1 \bmod p^{i+1}$: specifically, just modify the transformation in (3.1) to let $\mathbf{a}' = \mathbf{a} - r' \cdot (q/p^{j+i}) \cdot \mathbf{e}_1$, so that $b' = b = \langle \mathbf{a}', \mathbf{s} \rangle / q + e + (pr + r'(s'_1/p^i))/p^j$. This procedure works as long as p^{j+i} divides q , so we can recover $s_1 \bmod p^{e-j+1}$.

Using the above reductions and the Chinese remainder theorem, and letting j_i be the above minimal value of j for $p = p_i$ (of which at most ℓ of these are greater than 1), from $A_{\mathbf{s}, \alpha}$ we can recover \mathbf{s} modulo

$$P = \prod_i p_i^{e_i - (j_i - 1)} = q / \prod_i p_i^{j_i - 1} \geq q \cdot \left(\frac{\alpha'}{\omega(\sqrt{\log n})} \right)^\ell \geq q \cdot \alpha \cdot \omega(\sqrt{\log n}),$$

because $\alpha' < \omega(\sqrt{\log n})/p_i^{j_i - 1}$ for all i by definition of j_i and by hypothesis on α' . By applying the ‘shift’ transformation to $A_{\mathbf{s}, \alpha}$ we can assume that $\mathbf{s} = 0 \bmod P$. Now every $\langle \mathbf{a}, \mathbf{s}' \rangle / q$ is an integer multiple of $P/q \geq \alpha \cdot \omega(\sqrt{\log n})$, and since every noise term $e \leftarrow D_\alpha$ has magnitude $< (\alpha/2) \cdot \omega(\sqrt{\log n})$ with overwhelming probability, we can round the second component of every $(\mathbf{a}, b) \leftarrow A_{\mathbf{s}, \alpha}$ to the exact value of $\langle \mathbf{a}, \mathbf{s} \rangle / q \bmod 1$. From these we can solve for \mathbf{s} by Gaussian elimination, and we are done. \square

4 Primitive Lattices

At the heart of our new trapdoor generation algorithm (described in Section 5) is the construction of a very special family of lattices which have excellent geometric properties, and admit very fast and parallelizable decoding algorithms. The lattices are defined by means of what we call a *primitive matrix*. We say that a matrix $\mathbf{G} \in \mathbb{Z}_q^{n \times m}$ is primitive if its columns generate all of \mathbb{Z}_q^n , i.e., $\mathbf{G} \cdot \mathbb{Z}^m = \mathbb{Z}_q^n$.⁶

The main results of this section are summarized in the following theorem.

Theorem 4.1. *For any integers $q \geq 2$, $n \geq 1$, $k = \lceil \log_2 q \rceil$ and $m = nk$, there is a primitive matrix $\mathbf{G} \in \mathbb{Z}_q^{n \times m}$ such that*

- *The lattice $\Lambda^\perp(\mathbf{G})$ has a known basis $\mathbf{S} \in \mathbb{Z}^{m \times m}$ with $\|\tilde{\mathbf{S}}\| \leq \sqrt{5}$ and $\|\mathbf{S}\| \leq \max\{\sqrt{5}, \sqrt{k}\}$. Moreover, when $q = 2^k$, we have $\tilde{\mathbf{S}} = 2\mathbf{I}$ (so $\|\tilde{\mathbf{S}}\| = 2$) and $\|\mathbf{S}\| = \sqrt{5}$.*
- *Both \mathbf{G} and \mathbf{S} require little storage. In particular, they are sparse (with only $O(m)$ nonzero entries) and highly structured.*
- *Inverting $g_{\mathbf{G}}(\mathbf{s}, \mathbf{e}) := \mathbf{s}^t \mathbf{G} + \mathbf{e}^t \bmod q$ can be performed in quasilinear $O(n \cdot \log^c n)$ time for any $\mathbf{s} \in \mathbb{Z}_q^n$ and any $\mathbf{e} \in \mathcal{P}_{1/2}(q \cdot \mathbf{B}^{-t})$, where \mathbf{B} can denote either \mathbf{S} or $\tilde{\mathbf{S}}$. Moreover, the algorithm is perfectly parallelizable, running in polylogarithmic $O(\log^c n)$ time using n processors. When $q = 2^k$, the polylogarithmic term $O(\log^c n)$ is essentially just the cost of k additions and shifts on k -bit integers.*

⁶We do not say that \mathbf{G} is “full-rank,” because \mathbb{Z}_q is not a field when q is not prime, and the notion of rank for matrices over \mathbb{Z}_q is not well defined.

- *Preimage sampling for $f_{\mathbf{G}}(\mathbf{x}) = \mathbf{G}\mathbf{x} \bmod q$ with Gaussian parameter $s \geq \|\tilde{\mathbf{S}}\| \cdot \omega(\sqrt{\log n})$ can be performed in quasilinear $O(n \cdot \log^c n)$ time, or parallel polylogarithmic $O(\log^c n)$ time using n processors. When $q = 2^k$, the polylogarithmic term is essentially just the cost of k additions and shifts on k -bit integers, plus the (offline) generation of about m random integers drawn from $D_{\mathbb{Z},s}$.*

More generally, for any integer $b \geq 2$, all of the above statements hold with $k = \lceil \log_b q \rceil$, $\|\tilde{\mathbf{S}}\| \leq \sqrt{b^2 + 1}$, and $\|\mathbf{S}\| \leq \max\{\sqrt{b^2 + 1}, (b-1)\sqrt{k}\}$; and when $q = b^k$, we have $\tilde{\mathbf{S}} = b\mathbf{I}$ and $\|\mathbf{S}\| = \sqrt{b^2 + 1}$.

The rest of this section is dedicated to the proof of Theorem 4.1. In the process, we also make several important observations regarding the implementation of the inversion and sampling algorithms associated with \mathbf{G} , showing that our algorithms are not just asymptotically fast, but also quite practical.

Let $q \geq 2$ be an integer modulus and $k \geq 1$ be an integer dimension. Our construction starts with a *primitive vector* $\mathbf{g} \in \mathbb{Z}_q^k$, i.e., a vector such that $\gcd(g_1, \dots, g_k, q) = 1$. The vector \mathbf{g} defines a k -dimensional lattice $\Lambda^\perp(\mathbf{g}^t) \subset \mathbb{Z}^k$ having determinant $|\mathbb{Z}^k / \Lambda^\perp(\mathbf{g}^t)| = q$, because the residue classes of $\mathbb{Z}^k / \Lambda^\perp(\mathbf{g}^t)$ are in bijective correspondence with the possible values of $\langle \mathbf{g}, \mathbf{x} \rangle \bmod q$ for $\mathbf{x} \in \mathbb{Z}^k$, which cover all of \mathbb{Z}_q since \mathbf{g} is primitive. Concrete primitive vectors \mathbf{g} will be described in the next subsections. Notice that when $q = \text{poly}(n)$, we have $k = O(\log q) = O(\log n)$ and so $\Lambda^\perp(\mathbf{g}^t)$ is a very low-dimensional lattice. Let $\mathbf{S}_k \in \mathbb{Z}^{k \times k}$ be a basis of $\Lambda^\perp(\mathbf{g}^t)$, that is, $\mathbf{g}^t \cdot \mathbf{S}_k = \mathbf{0} \in \mathbb{Z}_q^{1 \times k}$ and $|\det(\mathbf{S}_k)| = q$.

The primitive vector \mathbf{g} and associated basis \mathbf{S}_k are used to define the parity-check matrix \mathbf{G} and basis $\mathbf{S} \in \mathbb{Z}_q$ as $\mathbf{G} := \mathbf{I}_n \otimes \mathbf{g}^t \in \mathbb{Z}_q^{n \times nk}$ and $\mathbf{S} := \mathbf{I}_n \otimes \mathbf{S}_k \in \mathbb{Z}^{nk \times nk}$. That is,

$$\mathbf{G} := \begin{bmatrix} \dots \mathbf{g}^t \dots & & & \\ & \dots \mathbf{g}^t \dots & & \\ & & \ddots & \\ & & & \dots \mathbf{g}^t \dots \end{bmatrix} \in \mathbb{Z}_q^{n \times nk}, \quad \mathbf{S} := \begin{bmatrix} \mathbf{S}_k & & & \\ & \mathbf{S}_k & & \\ & & \ddots & \\ & & & \mathbf{S}_k \end{bmatrix} \in \mathbb{Z}^{nk \times nk}.$$

Equivalently, \mathbf{G} , $\Lambda^\perp(\mathbf{G})$, and \mathbf{S} are the direct sums of n copies of \mathbf{g}^t , $\Lambda^\perp(\mathbf{g}^t)$, and \mathbf{S}_k , respectively. It follows that \mathbf{G} is a primitive matrix, the lattice $\Lambda^\perp(\mathbf{G}) \subset \mathbb{Z}^{nk}$ has determinant q^n , and \mathbf{S} is a basis for this lattice. It also follows (and is clear by inspection) that $\|\mathbf{S}\| = \|\mathbf{S}_k\|$ and $\|\tilde{\mathbf{S}}\| = \|\tilde{\mathbf{S}}_k\|$.

By this direct sum construction, it is immediate that inverting $g_{\mathbf{G}}(\mathbf{s}, \mathbf{e})$ and sampling preimages of $f_{\mathbf{G}}(\mathbf{x})$ can be accomplished by performing the same operations n times in parallel for $g_{\mathbf{g}^t}$ and $f_{\mathbf{g}^t}$ on the corresponding portions of the input, and concatenating the results. For preimage sampling, if each of the $f_{\mathbf{g}^t}$ preimages has Gaussian parameter $\sqrt{\Sigma}$, then by independence, their concatenation has parameter $\mathbf{I}_n \otimes \sqrt{\Sigma}$. Likewise, inverting $g_{\mathbf{G}}$ will succeed whenever all the n independent $g_{\mathbf{g}^t}$ -inversion subproblems are solved correctly.

In the next two subsections we study concrete instantiations of the primitive vector \mathbf{g} , and give optimized algorithms for inverting $g_{\mathbf{g}^t}$ and sampling preimages for $f_{\mathbf{g}^t}$. In both subsections, we consider primitive lattices $\Lambda^\perp(\mathbf{g}^t) \subset \mathbb{Z}^k$ defined by the vector

$$\mathbf{g}^t := [1 \quad 2 \quad 4 \quad \dots \quad 2^{k-1}] \in \mathbb{Z}_q^{1 \times k}, \quad k = \lceil \log_2 q \rceil, \quad (4.1)$$

whose entries form a geometrically increasing sequence. (We focus on powers of 2, but all our results trivially extend to other integer powers, or even mixed-integer products.) The only difference between the two subsections is in the form of the modulus q . We first study the case when the modulus $q = 2^k$ is a power of 2, which leads to especially simple and fast algorithms. Then we discuss how the results can be generalized to arbitrary moduli q . Notice that in both cases, the syndrome $\langle \mathbf{g}, \mathbf{x} \rangle \in \mathbb{Z}_q$ of a binary

vector $\mathbf{x} = (x_0, \dots, x_{k-1}) \in \{0, 1\}^k$ is just the positive integer with binary expansion \mathbf{x} . In general, for arbitrary $\mathbf{x} \in \mathbb{Z}^k$ the syndrome $\langle \mathbf{g}, \mathbf{x} \rangle \in \mathbb{Z}_q$ can be computed very efficiently by a sequence of k additions and binary shifts, and a single reduction modulo q , which is also trivial when $q = 2^k$ is a power of 2. The syndrome computation is also easily parallelizable, leading to $O(\log k) = O(\log \log n)$ computation time using $O(k) = O(\log n)$ processors.

4.1 Power-of-Two Modulus

Let $q = 2^k$ be a power of 2, and let \mathbf{g} be the geometric vector defined in Equation (4.1). Define the matrix

$$\mathbf{S}_k := \begin{bmatrix} 2 & & & & \\ -1 & 2 & & & \\ & -1 & \ddots & & \\ & & & 2 & \\ & & & -1 & 2 \end{bmatrix} \in \mathbb{Z}^{k \times k}.$$

This is a basis for $\Lambda^\perp(\mathbf{g}^t)$, because $\mathbf{g}^t \cdot \mathbf{S}_k = \mathbf{0} \bmod q$ and $\det(\mathbf{S}_k) = 2^k = q$. Clearly, all the basis vectors are short. Moreover, by orthogonalizing \mathbf{S}_k in reverse order, we have $\widetilde{\mathbf{S}}_k = 2 \cdot \mathbf{I}_k$. This construction is summarized in the following proposition. (It generalizes in the obvious way to any integer base, not just 2.)

Proposition 4.2. *For $q = 2^k$ and $\mathbf{g} = (1, 2, \dots, 2^{k-1}) \in \mathbb{Z}_q^k$, the lattice $\Lambda^\perp(\mathbf{g}^t)$ has a basis \mathbf{S} such that $\widetilde{\mathbf{S}} = 2\mathbf{I}$ and $\|\mathbf{S}\| \leq \sqrt{5}$. In particular, $\eta_\epsilon(\Lambda^\perp(\mathbf{g}^t)) \leq 2r = 2 \cdot \omega(\sqrt{\log n})$ for some $\epsilon(n) = \text{negl}(n)$.*

Using Proposition 4.2 and known generic algorithms [Bab85, Kle00, GPV08], it is possible to invert $g_{\mathbf{g}^t}(\mathbf{s}, \mathbf{e})$ correctly whenever $\mathbf{e} \in \mathcal{P}_{1/2}((q/2) \cdot \mathbf{I})$, and sample preimages under $f_{\mathbf{g}^t}$ with Gaussian parameter $s \geq 2r = 2 \cdot \omega(\sqrt{\log n})$. In what follows we show how the special structure of the basis \mathbf{S} leads to simpler, faster, and more practical solutions to these general lattice problems.

Inversion. Here we show how to efficiently find an unknown scalar $s \in \mathbb{Z}_q$ given $\mathbf{b}^t = [b_0, b_1, \dots, b_{k-1}] = s \cdot \mathbf{g}^t + \mathbf{e}^t = [s + e_0, 2s + e_1, \dots, 2^{k-1}s + e_{k-1}] \bmod q$, where $\mathbf{e} \in \mathbb{Z}^k$ is a short error vector.

An iterative algorithm works by recovering the binary digits $s_0, s_1, \dots, s_{k-1} \in \{0, 1\}$ of $s \in \mathbb{Z}_q$, from least to most significant, as follows: first, determine s_0 by testing whether

$$b_{k-1} = 2^{k-1}s + e_{k-1} = (q/2)s_0 + e_{k-1} \bmod q$$

is closer to 0 or to $q/2$ (modulo q). Then recover s_1 from $b_{k-2} = 2^{k-2}s + e_{k-2} = 2^{k-1}s_1 + 2^{k-2}s_0 + e_{k-2} \bmod q$, by subtracting $2^{k-2}s_0$ and testing proximity to 0 or $q/2$, etc. It is easy to see that the algorithm produces correct output if every $e_i \in [-\frac{q}{4}, \frac{q}{4}]$, i.e., if $\mathbf{e} \in \mathcal{P}_{1/2}(q \cdot \mathbf{I}_k/2) = \mathcal{P}_{1/2}(q \cdot (\widetilde{\mathbf{S}}_k)^{-t})$. It can also be seen that this algorithm is exactly Babai's "nearest-plane" algorithm [Bab85], specialized to the scaled dual $q(\mathbf{S}_k)^{-t}$ of the basis \mathbf{S}_k of $\Lambda^\perp(\mathbf{g}^t)$, which is a basis for $\Lambda(\mathbf{g})$.

Formally, the iterative algorithm is: given a vector $\mathbf{b}^t = [b_0, \dots, b_{k-1}] \in \mathbb{Z}_q^{1 \times k}$, initialize $s \leftarrow 0$.

1. For $i = k-1, \dots, 0$: let $s \leftarrow s + 2^{k-1-i} \cdot [b_i - 2^i \cdot s \notin [-\frac{q}{4}, \frac{q}{4}] \bmod q]$, where $[E] = 1$ if expression E is true, and 0 otherwise. Also let $e_i \leftarrow b_i - 2^i \cdot s \in [-\frac{q}{4}, \frac{q}{4}]$.
2. Output $s \in \mathbb{Z}_q$ and $\mathbf{e} = (e_0, \dots, e_{k-1}) \in [-\frac{q}{4}, \frac{q}{4}]^k \subset \mathbb{Z}^k$.

Note that for $x \in \{0, \dots, q-1\}$ with binary representation $(x_{k-1}x_{k-2} \dots x_0)_2$, we have

$$\left\lceil x \right\rceil \left[-\frac{q}{4}, \frac{q}{4} \right) \bmod q = x_{k-1} \oplus x_{k-2}.$$

There is also a non-iterative approach to decoding using a lookup table, and a hybrid approach between the two extremes. Notice that rounding each entry b_i of \mathbf{b} to the nearest multiple of 2^i (modulo q , breaking ties upward) before running the above algorithm does not change the value of s that is computed. This lets us precompute a lookup table that maps the $2^{k(k+1)/2} = q^{O(\lg q)}$ possible rounded values of \mathbf{b} to the correct values of s . The size of this table grows very rapidly for $k > 3$, but in this case we can do better if we assume slightly smaller error terms $e_i \in [-\frac{q}{8}, \frac{q}{8})$: simply round each b_i to the nearest multiple of $\max\{\frac{q}{8}, 2^i\}$, thus producing one of exactly $8^{k-1} = q^3/8$ possible results, whose solutions can be stored in a lookup table. Note that the result is correct, because in each coordinate the total error introduced by e_i and rounding to a multiple of $\frac{q}{8}$ is in the range $[-\frac{q}{4}, \frac{q}{4})$. A hybrid approach combining the iterative algorithm with table lookups of ℓ bits of s at a time is potentially the most efficient option in practice, and is easy to devise from the above discussion.

Gaussian sampling. We now consider the preimage sampling problem for function $f_{\mathbf{g}^t}$, i.e., the task of Gaussian sampling over a desired coset of $\Lambda^\perp(\mathbf{g}^t)$. More specifically, we want to sample a vector from the set $\Lambda_u^\perp(\mathbf{g}^t) = \{\mathbf{x} \in \mathbb{Z}^k : \langle \mathbf{g}, \mathbf{x} \rangle = u \bmod q\}$ for a desired syndrome $u \in \mathbb{Z}_q$, with probability proportional to $\rho_s(\mathbf{x})$. We wish to do so for any fixed Gaussian parameter $s \geq \|\widetilde{\mathbf{S}}_k\| \cdot r = 2 \cdot \omega(\sqrt{\log n})$, which is an optimal bound on the smoothing parameter of $\Lambda^\perp(\mathbf{G})$.

As with inversion, there are two main approaches to Gaussian sampling, which are actually opposite extremes on a spectrum of storage/parallelism trade-offs. The first approach is essentially to precompute and store many independent samples $\mathbf{x} \leftarrow D_{\mathbb{Z}^k, s}$, ‘bucketing’ them based on the value of $\langle \mathbf{g}, \mathbf{x} \rangle \in \mathbb{Z}_q$ until there is at least one sample per bucket. Because each $\langle \mathbf{g}, \mathbf{x} \rangle$ is statistically close to uniform over \mathbb{Z}_q (by the smoothing parameter bound for $\Lambda^\perp(\mathbf{g}^t)$), a coupon-collecting argument implies that we need to generate about $q \log q$ samples to occupy every bucket. The online part of the sampling algorithm for $\Lambda^\perp(\mathbf{g}^t)$ is trivial, merely taking a fresh \mathbf{x} from the appropriate bucket. The downside is that the storage and precomputation requirements are rather high: in many applications, q (while polynomial in the security parameter) can be in the many thousands or more.

The second approach exploits the niceness of the orthogonalized basis $\widetilde{\mathbf{S}}_k = 2\mathbf{I}_k$. Using this basis, the randomized nearest-plane algorithm of [Kle00, GPV08] becomes very simple and efficient, and is equivalent to the following: given a syndrome $u \in \{0, \dots, q-1\}$ (viewed as an integer),

1. For $i = 0, \dots, k-1$: choose $x_i \leftarrow D_{2\mathbb{Z}+u, s}$ and let $u \leftarrow (u - x_i)/2 \in \mathbb{Z}$.
2. Output $\mathbf{x} = (x_0, \dots, x_{k-1})$.

Observe that every Gaussian x_i in the above algorithm is chosen from one of only two possible cosets of $2\mathbb{Z}$, determined by the least significant bit of u at that moment. Therefore, we may precompute and store several independent Gaussian samples from each of $2\mathbb{Z}$ and $2\mathbb{Z}+1$, and consume one per iteration when executing the algorithm. (As above, the individual samples may be generated by choosing several $x \leftarrow D_{\mathbb{Z}, s}$ and bucketing each one according to its least-significant bit.) Such presampling makes the algorithm deterministic during its online phase, and because there are only two cosets, there is almost no wasted storage or precomputation. Notice, however, that this algorithm requires $k = \lg(q)$ sequential iterations.

Between the extremes of the two algorithms described above, there is a hybrid algorithm that chooses $\ell \geq 1$ entries of \mathbf{x} at a time. (For simplicity, we assume that ℓ divides k exactly, though this is not

strictly necessary.) Let $\mathbf{h}^t = [1, 2, \dots, 2^{\ell-1}] \in \mathbb{Z}_2^{1 \times \ell}$ be a parity-check matrix defining the 2^ℓ -ary lattice $\Lambda^\perp(\mathbf{h}^t) \subseteq \mathbb{Z}^\ell$, and observe that $\mathbf{g}^t = [\mathbf{h}^t, 2^\ell \cdot \mathbf{h}^t, \dots, 2^{k-\ell} \cdot \mathbf{h}^t]$. The hybrid algorithm then works as follows:

1. For $i = 0, \dots, k/\ell - 1$, choose $(x_{i\ell}, \dots, x_{(i+1)\ell-1}) \leftarrow D_{\Lambda_{u \bmod 2^\ell}^\perp(\mathbf{h}^t), s}$ and let $u \leftarrow (u - x)/2^\ell$, where $x = \sum_{j=0}^{\ell-1} x_{i\ell+j} \cdot 2^j \in \mathbb{Z}$.
2. Output $\mathbf{x} = (x_0, \dots, x_{k-1})$.

As above, we can precompute samples $\mathbf{x} \leftarrow D_{\mathbb{Z}^\ell, s}$ and store them in a lookup table having 2^ℓ buckets, indexed by the value $\langle \mathbf{h}, \mathbf{x} \rangle \in \mathbb{Z}_{2^\ell}$, thereby making the algorithm deterministic in its online phase.

4.2 Arbitrary Modulus

For a modulus q that is not a power of 2, most of the above ideas still work, with slight adaptations. Let $k = \lceil \lg(q) \rceil$, so $q < 2^k$. As above, define $\mathbf{g}^t := [1, 2, \dots, 2^{k-1}] \in \mathbb{Z}_q^{1 \times k}$, but now define the matrix

$$\mathbf{S}_k := \begin{bmatrix} 2 & & & & q_0 \\ -1 & 2 & & & q_1 \\ & -1 & & & q_2 \\ & & \ddots & & \vdots \\ & & & 2 & q_{k-2} \\ & & & -1 & q_{k-1} \end{bmatrix} \in \mathbb{Z}^{k \times k}$$

where $(q_0, \dots, q_{k-1}) \in \{0, 1\}^k$ is the binary expansion of $q = \sum_i 2^i \cdot q_i$. Again, \mathbf{S} is a basis of $\Lambda^\perp(\mathbf{g}^t)$ because $\mathbf{g}^t \cdot \mathbf{S}_k = \mathbf{0} \bmod q$, and $\det(\mathbf{S}_k) = q$. Moreover, the basis vectors have squared length $\|\mathbf{s}_i\|^2 = 5$ for $i < k$ and $\|\mathbf{s}_k\|^2 = \sum_i q_i \leq k$. The next lemma shows that \mathbf{S}_k also has a good Gram-Schmidt orthogonalization.

Lemma 4.3. *With $\mathbf{S} = \mathbf{S}_k$ defined as above and orthogonalized in forward order, we have $\|\tilde{\mathbf{s}}_i\|^2 = \frac{4-4^{-i}}{1-4^{-i}} \in (4, 5]$ for $1 \leq i < k$, and $\|\tilde{\mathbf{s}}_k\|^2 = \frac{3q^2}{4^k-1} < 3$.*

Proof. Notice that the vectors $\mathbf{s}_1, \dots, \mathbf{s}_{k-1}$ are all orthogonal to $\mathbf{g}_k = (1, 2, 4, \dots, 2^{k-1}) \in \mathbb{Z}^k$. Thus, the orthogonal component of \mathbf{s}_k has squared length

$$\|\tilde{\mathbf{s}}_k\|^2 = \frac{\langle \mathbf{s}_k, \mathbf{g}_k \rangle^2}{\|\mathbf{g}_k\|^2} = \frac{q^2}{\sum_{j < k} 4^j} = \frac{3q^2}{4^k - 1}.$$

Similarly, the squared length of $\tilde{\mathbf{s}}_i$ for $i < k$ can be computed as

$$\|\tilde{\mathbf{s}}_i\|^2 = 1 + \frac{4^i}{\sum_{j < i} 4^j} = \frac{4 - 4^{-i}}{1 - 4^{-i}}. \quad \square$$

This concludes the description and analysis of the primitive lattice $\Lambda^\perp(\mathbf{g}^t)$ when q is not a power of 2. Specialized inversion algorithms can also be adapted as well, but some care is needed. Of course, since the lattice dimension $k = O(\log n)$ is very small, one could simply use the general methods of [Bab85, Kle00, GPV08, Pei10] without worrying too much about optimizations, and satisfy all the claims made in Theorem 4.1. Below we briefly discuss alternatives for Gaussian sampling.

The offline ‘bucketing’ approach to Gaussian sampling works without any modification for arbitrary modulus, with just slightly larger Gaussian parameter $s \geq \sqrt{5} \cdot r$, because it relies only on the smoothing parameter bound of $\eta_\epsilon(\Lambda^\perp(\mathbf{g}^t)) \leq \|\widetilde{\mathbf{S}}_k\| \cdot \omega(\sqrt{\log n})$ and the fact that the number of buckets is q . The randomized nearest-plane approach to sampling does not admit a specialization as simple as the one we have described for $q = 2^k$. The reason is that while the basis \mathbf{S} is sparse, its orthogonalization $\widetilde{\mathbf{S}}$ is not sparse in general. (This is in contrast to the case when $q = 2^k$, for which orthogonalizing in reverse order leads to the sparse matrix $\widetilde{\mathbf{S}} = 2\mathbf{I}$.) Still, $\widetilde{\mathbf{S}}$ is “almost triangular,” in the sense that the off-diagonal entries decrease geometrically as one moves away from the diagonal. This may allow for optimizing the sampling algorithm by performing “truncated” scalar product computations, and still obtain an almost-Gaussian distribution on the resulting samples. An interesting alternative is to use a hybrid approach, where one first performs a single iteration of randomized nearest-plane algorithm to take care of the last basis vector \mathbf{s}_k , and then performs some variant of the convolution algorithm from [Pei10] to deal with the first $k-1$ basis vectors $[\mathbf{s}_1, \dots, \mathbf{s}_{k-1}]$, which have very small lengths and singular values. Notice that the orthogonalized component of the last vector \mathbf{s}_k is simply a scalar multiple of the primitive vector \mathbf{g} , so the scalar product $\langle \mathbf{s}_k, \mathbf{t} \rangle$ (for any vector \mathbf{t} with syndrome $u = \langle \mathbf{g}, \mathbf{t} \rangle$) can be immediately computed from u as u/q (see Lemma 4.3).

4.3 The Ring Setting

The above constructions and algorithms all transfer easily to compact lattices defined over polynomial rings (i.e., number rings), as used in the representative works [Mic02, PR06, LM06, LPR10]. A commonly used example is the cyclotomic ring $R = \mathbb{Z}[x]/(\Phi_m(x))$ where $\Phi_m(x)$ denotes the m th cyclotomic polynomial, which is a monic, degree- $\varphi(m)$, irreducible polynomial whose zeros are all the *primitive* m th roots of unity in \mathbb{C} . The ring R is a \mathbb{Z} -module of rank n , i.e., it is generated as the additive integer combinations of the “power basis” elements $1, x, x^2, \dots, x^{\varphi(m)-1}$. We let $R_q = R/qR$, the ring modulo the ideal generated by an integer q . For geometric concepts like error vectors and Gaussian distributions, it is usually nicest to work with the “canonical embedding” of R , which roughly (but not exactly) corresponds with the “coefficient embedding,” which just considers the vector of coefficients relative to the power basis.

Let $\mathbf{g} \in R_q^k$ be a primitive vector modulo q , i.e., one for which the ideal generated by q, g_1, \dots, g_k is the full ring R . As above, the vector \mathbf{g} defines functions $f_{\mathbf{g}^t} : R^k \rightarrow R_q$ and $g_{\mathbf{g}^t} : R_q \times R^k \rightarrow R_q^{1 \times k}$, defined as $f_{\mathbf{g}^t}(\mathbf{x}) = \langle \mathbf{g}, \mathbf{x} \rangle = \sum_{i=1}^k g_i \cdot x_i \bmod q$ and $g_{\mathbf{g}^t}(s, \mathbf{e}) = s \cdot \mathbf{g}^t + \mathbf{e}^t \bmod q$, and the related R -module

$$qR^k \subseteq \Lambda^\perp(\mathbf{g}^t) := \{\mathbf{x} \in R^k : f_{\mathbf{g}^t}(\mathbf{x}) = \langle \mathbf{g}, \mathbf{x} \rangle = 0 \bmod q\} \subsetneq R^k,$$

which has index (determinant) $q^n = |R_q|$ as an additive subgroup of R^k because \mathbf{g} is primitive. Concretely, we can use the exact same primitive vector $\mathbf{g}^t = [1, 2, \dots, 2^{k-1}] \in R_q^k$ as in Equation (4.1), interpreting its entries in the ring R_q rather than \mathbb{Z}_q .

Inversion and preimage sampling algorithms for $g_{\mathbf{g}^t}$ and $f_{\mathbf{g}^t}$ (respectively) are relatively straightforward to obtain, by adapting the basic approaches from the previous subsections. These algorithms are simplest when the power basis elements $1, x, x^2, \dots, x^{\varphi(m)-1}$ are *orthogonal* under the canonical embedding (which is the case exactly when m is a power of 2, and hence $\Phi_m(x) = x^{m/2} + 1$), because the inversion operations reduce to parallel operations relative to each of the power basis elements. We defer the details to the full version.

5 Trapdoor Generation and Operations

In this section we describe our new trapdoor generation, inversion and sampling algorithms for hard random lattices. Recall that these are lattices $\Lambda^\perp(\mathbf{A})$ defined by an (almost) uniformly random matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, and that the standard notion of a “strong” trapdoor for these lattices (put forward in [GPV08] and used in a large number of subsequent applications) is a short lattice basis $\mathbf{S} \in \mathbb{Z}^{m \times m}$ for $\Lambda^\perp(\mathbf{A})$. There are several measures of quality for the trapdoor \mathbf{S} , the most common ones being (in nondecreasing order): the maximal Gram-Schmidt length $\|\tilde{\mathbf{S}}\|$; the maximal Euclidean length $\|\mathbf{S}\|$; and the maximal singular value $s_1(\mathbf{S})$. Algorithms for generating random lattices together with high-quality trapdoor bases are given in [Ajt99, AP09]. In this section we give much simpler, faster and tighter algorithms to generate a hard random lattice with a trapdoor, and to use a trapdoor for performing standard tasks like inverting the LWE function $g_{\mathbf{A}}$ and sampling preimages for the SIS function $f_{\mathbf{A}}$. We also give a new, simple algorithm for delegating a trapdoor, i.e., using a trapdoor for \mathbf{A} to obtain one for a matrix $[\mathbf{A} \mid \mathbf{A}']$ that extends \mathbf{A} , in a secure and non-reversible way.

The following theorem summarizes the main results of this section. Here we state just one typical instantiation with only asymptotic bounds. More general results and exact bounds are presented throughout the section.

Theorem 5.1. *There is an efficient randomized algorithm $\text{GenTrap}(1^n, 1^m, q)$ that, given any integers $n \geq 1$, $q \geq 2$, and sufficiently large $m = O(n \log q)$, outputs a parity-check matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ and a ‘trapdoor’ \mathbf{R} such that the distribution of \mathbf{A} is $\text{negl}(n)$ -far from uniform. Moreover, there are efficient algorithms Invert and SampleD that with overwhelming probability over all random choices, do the following:*

- For $\mathbf{b}^t = \mathbf{s}^t \mathbf{A} + \mathbf{e}^t$, where $\mathbf{s} \in \mathbb{Z}_q^n$ is arbitrary and either $\|\mathbf{e}\| < q/O(\sqrt{n \log q})$ or $\mathbf{e} \leftarrow D_{\mathbb{Z}^m, \alpha q}$ for $1/\alpha \geq \sqrt{n \log q} \cdot \omega(\sqrt{\log n})$, the deterministic algorithm $\text{Invert}(\mathbf{R}, \mathbf{A}, \mathbf{b})$ outputs \mathbf{s} and \mathbf{e} .
- For any $\mathbf{u} \in \mathbb{Z}_q^n$ and large enough $s = O(\sqrt{n \log q})$, the randomized algorithm $\text{SampleD}(\mathbf{R}, \mathbf{A}, \mathbf{u}, s)$ samples from a distribution within $\text{negl}(n)$ statistical distance of $D_{\Lambda_{\mathbf{u}}^\perp(\mathbf{A}), s \cdot \omega(\sqrt{\log n})}$.

Throughout this section, we let $\mathbf{G} \in \mathbb{Z}_q^{n \times w}$ denote some fixed primitive matrix that admits efficient inversion and preimage sampling algorithms, as described in Theorem 4.1. (Recall that typically, $w = n \lceil \log q \rceil$ for some appropriate base of the logarithm.) All our algorithms and efficiency improvements are based on the primitive matrix \mathbf{G} and associated algorithms described in Section 4, and a new notion of trapdoor that we define next.

5.1 A New Trapdoor Notion

We begin by defining the new notion of trapdoor, establish some of its most important properties, and give a simple and efficient algorithm for generating hard random lattices together with high-quality trapdoors.

Definition 5.2. Let $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ and $\mathbf{G} \in \mathbb{Z}_q^{n \times w}$ be matrices with $m \geq w \geq n$. A \mathbf{G} -trapdoor for \mathbf{A} is a matrix $\mathbf{R} \in \mathbb{Z}^{(m-w) \times w}$ such that $\mathbf{A} \begin{bmatrix} \mathbf{R} \\ \mathbf{I} \end{bmatrix} = \mathbf{H} \mathbf{G}$ for some invertible matrix $\mathbf{H} \in \mathbb{Z}_q^{n \times n}$. We refer to \mathbf{H} as the *tag* or *label* of the trapdoor. The *quality* of the trapdoor is measured by its largest singular value $s_1(\mathbf{R})$.

We remark that, by definition of \mathbf{G} -trapdoor, if \mathbf{G} is a primitive matrix and \mathbf{A} admits a \mathbf{G} trapdoor, then \mathbf{A} is primitive as well. In particular, $\det(\Lambda^\perp(\mathbf{A})) = q^n$. Since the primitive matrix \mathbf{G} is typically fixed and public, we usually omit references to it, and refer to \mathbf{G} -trapdoors simply as trapdoors. We remark that since

\mathbf{G} is primitive, the tag \mathbf{H} in the above definition is uniquely determined by (and efficiently computable from) \mathbf{A} and the trapdoor \mathbf{R} .

The following lemma says that a good basis for $\Lambda^\perp(\mathbf{A})$ may be obtained from knowledge of \mathbf{R} . We do not use the lemma anywhere in the rest of the paper, but include it here primarily to show that our new definition of trapdoor is at least as powerful as the traditional one of a short basis. Our algorithms for Gaussian sampling and LWE inversion do not need a full basis, and make direct (and more efficient) use of our new notion of trapdoor.

Lemma 5.3. *Let $\mathbf{S} \in \mathbb{Z}^{w \times w}$ be any basis for $\Lambda^\perp(\mathbf{G})$. Let $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ have trapdoor $\mathbf{R} \in \mathbb{Z}^{(m-w) \times w}$ with tag $\mathbf{H} \in \mathbb{Z}_q^{n \times n}$. Then the lattice $\Lambda^\perp(\mathbf{A})$ is generated by the basis*

$$\mathbf{S}_\mathbf{A} = \begin{bmatrix} \mathbf{I} & \mathbf{R} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{W} & \mathbf{S} \end{bmatrix},$$

where $\mathbf{W} \in \mathbb{Z}^{w \times \tilde{m}}$ is an arbitrary solution to $\mathbf{G}\mathbf{W} = -\mathbf{H}^{-1}\mathbf{A}[\mathbf{I} \mid \mathbf{0}]^T \pmod{q}$. Moreover, the basis $\mathbf{S}_\mathbf{A}$ satisfies $\|\widetilde{\mathbf{S}}_\mathbf{A}\| \leq s_1(\begin{bmatrix} \mathbf{I} & \mathbf{R} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}) \cdot \|\widetilde{\mathbf{S}}\| \leq (s_1(\mathbf{R}) + 1) \cdot \|\widetilde{\mathbf{S}}\|$, when $\mathbf{S}_\mathbf{A}$ is orthogonalized in suitable order.

Proof. It is immediate to check that $\mathbf{A} \cdot \mathbf{S}_\mathbf{A} = \mathbf{0} \pmod{q}$, so $\mathbf{S}_\mathbf{A}$ generates a sublattice of $\Lambda^\perp(\mathbf{A})$. In fact, it generates the entire lattice because $\det(\mathbf{S}_\mathbf{A}) = \det(\mathbf{S}) = q^n = \det(\Lambda^\perp(\mathbf{A}))$.

The bound on $\|\widetilde{\mathbf{S}}_\mathbf{A}\|$ follows by simple linear algebra. Recall by Item 3 of Lemma 2.1 that $\|\widetilde{\mathbf{B}}\| = \|\widetilde{\mathbf{S}}\|$ when the columns of $\mathbf{B} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{W} & \mathbf{S} \end{bmatrix}$ are reordered appropriately. So it suffices to show that $\|\mathbf{T}\mathbf{B}\| \leq s_1(\mathbf{T}) \cdot \|\widetilde{\mathbf{B}}\|$ for any \mathbf{T}, \mathbf{B} . Let $\mathbf{B} = \mathbf{Q}\mathbf{D}\mathbf{U}$ and $\mathbf{T}\mathbf{B} = \mathbf{Q}'\mathbf{D}'\mathbf{U}'$ be Gram-Schmidt decompositions of \mathbf{B} and $\mathbf{T}\mathbf{B}$, respectively, with \mathbf{Q}, \mathbf{Q}' orthogonal, \mathbf{D}, \mathbf{D}' diagonal with nonnegative entries, and \mathbf{U}, \mathbf{U}' upper unitriangular. We have

$$\mathbf{T}\mathbf{Q}\mathbf{D}\mathbf{U} = \mathbf{Q}'\mathbf{D}'\mathbf{U}' \implies \mathbf{T}'\mathbf{D} = \mathbf{D}'\mathbf{U}'',$$

where $\mathbf{T} = \mathbf{Q}'\mathbf{T}'\mathbf{Q}^{-1} \implies s_1(\mathbf{T}') = s_1(\mathbf{T})$, and \mathbf{U}'' is upper unitriangular because such matrices form a multiplicative group. Now every row of $\mathbf{T}'\mathbf{D}$ has Euclidean norm at most $s_1(\mathbf{T}) \cdot \|\mathbf{D}\| = s_1(\mathbf{T}) \cdot \|\widetilde{\mathbf{B}}\|$, while the i th row of $\mathbf{D}'\mathbf{U}''$ has norm at least $d'_{i,i}$, the i th diagonal of \mathbf{D}' . We conclude that $\|\mathbf{T}\mathbf{B}\| = \|\mathbf{D}\| \leq s_1(\mathbf{T}) \cdot \|\widetilde{\mathbf{B}}\|$, as desired. \square

We also make the following simple but useful observations:

- The rows of $\begin{bmatrix} \mathbf{R} \\ \mathbf{I} \end{bmatrix}$ in Definition 5.2 can appear in any order, since this just induces a permutation of \mathbf{A} 's columns.
- If \mathbf{R} is a trapdoor for \mathbf{A} , then it can be made into an equally good trapdoor for any extension $[\mathbf{A} \mid \mathbf{B}]$, by padding \mathbf{R} with zero rows; this leaves $s_1(\mathbf{R})$ unchanged.
- If \mathbf{R} is a trapdoor for \mathbf{A} with tag \mathbf{H} , then \mathbf{R} is also a trapdoor for $\mathbf{A}' = \mathbf{A} - [\mathbf{0} \mid \mathbf{H}'\mathbf{G}]$ with tag $(\mathbf{H} - \mathbf{H}')$ for any $\mathbf{H}' \in \mathbb{Z}_q^{n \times n}$, as long as $(\mathbf{H} - \mathbf{H}')$ is invertible modulo q . This is the main idea behind the compact IBE of [ABB10a], and can be used to give a family of “tag-based” trapdoor functions [KMO10]. In Section 6 we give explicit families of matrices \mathbf{H} having suitable properties for applications.

5.2 Trapdoor Generation

We now give an algorithm to generate a (pseudo)random matrix \mathbf{A} together with a \mathbf{G} -trapdoor. The algorithm is straightforward, and in fact it can be easily derived from the definition of \mathbf{G} -trapdoor itself. A random

lattice is built by first extending the primitive matrix \mathbf{G} into a semi-random matrix $\mathbf{A}' = [\bar{\mathbf{A}} \mid \mathbf{H}\mathbf{G}]$ (where $\bar{\mathbf{A}} \in \mathbb{Z}_q^{n \times \bar{m}}$ is chosen at random, and $\mathbf{H} \in \mathbb{Z}_q^{n \times n}$ is the desired tag), and then applying a random transformation $\mathbf{T} = \begin{bmatrix} \mathbf{I} & \mathbf{R} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \in \mathbb{Z}^{m \times m}$ to the semi-random lattice $\Lambda^\perp(\mathbf{A}')$. Since \mathbf{T} is unimodular with inverse $\mathbf{T}^{-1} = \begin{bmatrix} \mathbf{I} & -\mathbf{R} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}$, by Lemma 2.1 this yields the lattice $\mathbf{T} \cdot \Lambda^\perp(\mathbf{A}') = \Lambda^\perp(\mathbf{A}' \cdot \mathbf{T}^{-1})$ associated with the parity-check matrix $\mathbf{A} = \mathbf{A}' \cdot \mathbf{T}^{-1} = [\bar{\mathbf{A}} \mid \mathbf{H}\mathbf{G} - \bar{\mathbf{A}}\mathbf{R}]$. Moreover, the distribution of \mathbf{A} is close to uniform (either statistically, or computationally) as long as the distribution of $[\bar{\mathbf{A}} \mid \mathbf{0}]\mathbf{T}^{-1} = [\bar{\mathbf{A}} \mid -\bar{\mathbf{A}}\mathbf{R}]$ is. For details, see Algorithm 1, whose correctness is immediate.

Algorithm 1 Efficient algorithm $\text{GenTrap}^\mathcal{D}(\bar{\mathbf{A}}, \mathbf{H})$ for generating a parity-check matrix \mathbf{A} with trapdoor \mathbf{R} .

Input: Matrix $\bar{\mathbf{A}} \in \mathbb{Z}_q^{n \times \bar{m}}$ for some $\bar{m} \geq 1$, invertible matrix $\mathbf{H} \in \mathbb{Z}_q^{n \times n}$, and distribution \mathcal{D} over $\mathbb{Z}^{\bar{m} \times w}$.

(If no particular $\bar{\mathbf{A}}, \mathbf{H}$ are given as input, then the algorithm may choose them itself, e.g., picking $\bar{\mathbf{A}} \in \mathbb{Z}_q^{n \times \bar{m}}$ uniformly at random, and setting $\mathbf{H} = \mathbf{I}$.)

Output: A parity-check matrix $\mathbf{A} = [\bar{\mathbf{A}} \mid \mathbf{A}_1] \in \mathbb{Z}_q^{n \times m}$, where $m = \bar{m} + w$, and trapdoor \mathbf{R} with tag \mathbf{H} .

- 1: Choose a matrix $\mathbf{R} \in \mathbb{Z}^{\bar{m} \times w}$ from distribution \mathcal{D} .
 - 2: Output $\mathbf{A} = [\bar{\mathbf{A}} \mid \mathbf{H}\mathbf{G} - \bar{\mathbf{A}}\mathbf{R}] \in \mathbb{Z}_q^{n \times m}$ and trapdoor $\mathbf{R} \in \mathbb{Z}^{\bar{m} \times w}$.
-

We next describe two types of GenTrap instantiations. The first type generates a trapdoor \mathbf{R} for a statistically near-uniform output matrix \mathbf{A} using dimension $\bar{m} \approx n \log q$ or less (there is a trade-off between \bar{m} and the trapdoor quality $s_1(\mathbf{R})$). The second types generates a computationally *pseudorandom* \mathbf{A} (under the LWE assumption) using dimension $\bar{m} = 2n$; this pseudorandom construction is the first of its kind in the literature. Certain applications allow for an optimization that decreases \bar{m} by an additive n term; this is most significant in the computationally secure construction because it yields $\bar{m} = n$.

Statistical instantiation. This instantiation works for any parameter \bar{m} and distribution \mathcal{D} over $\mathbb{Z}^{\bar{m} \times w}$ having the following two properties:

1. *Subgaussianity:* \mathcal{D} is subgaussian with some parameter $s > 0$ (or δ -subgaussian for some small δ). This implies by Lemma 2.9 that $\mathbf{R} \leftarrow \mathcal{D}$ has $s_1(\mathbf{R}) = s \cdot O(\sqrt{\bar{m}} + \sqrt{w})$, except with probability $2^{-\Omega(\bar{m}+w)}$. (Recall that the constant factor hidden in the $O(\cdot)$ expression is $\approx 1/\sqrt{2\pi}$.)
2. *Regularity:* for $\bar{\mathbf{A}} \leftarrow \mathbb{Z}_q^{n \times \bar{m}}$ and $\mathbf{R} \leftarrow \mathcal{D}$, $\mathbf{A} = [\bar{\mathbf{A}} \mid \bar{\mathbf{A}}\mathbf{R}]$ is δ -uniform for some $\delta = \text{negl}(n)$.

In fact, there is no loss in security if $\bar{\mathbf{A}}$ contains an identity matrix \mathbf{I} as a submatrix and is otherwise uniform, since this corresponds with the Hermite normal form of the SIS and LWE problems. See, e.g., [MR09, Section 5] for further details.

For example, let $\mathcal{D} = \mathcal{P}^{\bar{m} \times w}$ where \mathcal{P} is the distribution over \mathbb{Z} that outputs 0 with probability $1/2$, and ± 1 each with probability $1/4$. Then \mathcal{P} (and hence \mathcal{D}) is 0-subgaussian with parameter $\sqrt{2\pi}$, and satisfies the regularity condition (for any q) for $\delta \leq \frac{w}{2} \sqrt{q^n/2^{\bar{m}}}$, by a version of the leftover hash lemma (see, e.g., [AP09, Section 2.2.1]). Therefore, we can use any $\bar{m} \geq n \lg q + 2 \lg \frac{w}{2\delta}$.

As another important example, let $\mathcal{D} = D_{\mathbb{Z}, s}^{\bar{m} \times w}$ be a discrete Gaussian distribution for some $s \geq \eta_\epsilon(\mathbb{Z})$ and $\epsilon = \text{negl}(n)$. Then \mathcal{D} is 0-subgaussian with parameter s by Lemma 2.8, and satisfies the regularity condition when \bar{m} satisfies the bound (2.2) from Lemma 2.4. For example, letting $s = 2\eta_\epsilon(\mathbb{Z})$ we can use any $\bar{m} = n \lg q + \omega(\log n)$. (Other tradeoffs between s and \bar{m} are possible, potentially using a different choice of \mathbf{G} , and more exact bounds on the error probabilities can be worked out from the lemma statements.) Moreover, by Lemmas 2.4 and 2.8 we have that with overwhelming probability over the choice of $\bar{\mathbf{A}}$, the

conditional distribution of \mathbf{R} given $\mathbf{A} = [\bar{\mathbf{A}} \mid \bar{\mathbf{A}}\mathbf{R}]$ is $\text{negl}(n)$ -subgaussian with parameter s . We will use this fact in some of our applications in Section 6.

Computational instantiation. Let $\bar{\mathbf{A}} = [\mathbf{I} \mid \hat{\mathbf{A}}] \in \mathbb{Z}_q^{n \times \bar{m}}$ for $\bar{m} = 2n$, and let $\mathcal{D} = D_{\mathbb{Z}, s}^{\bar{m} \times w}$ for some $s = \alpha q$, where $\alpha > 0$ is an LWE relative error rate (and typically $\alpha q > \sqrt{n}$). Clearly, \mathcal{D} is 0-subgaussian with parameter αq . Also, $[\bar{\mathbf{A}} \mid \bar{\mathbf{A}}\mathbf{R} = \hat{\mathbf{A}}\mathbf{R}_2 + \mathbf{R}_1]$ for $\mathbf{R} = \begin{bmatrix} \mathbf{R}_1 \\ \mathbf{R}_2 \end{bmatrix} \leftarrow \mathcal{D}$ is exactly an instance of decision-LWE $_{n, q, \alpha}$ (in its normal form), and hence is pseudorandom (ignoring the identity submatrix) assuming that the problem is hard.

Further optimizations. If an application only uses a single tag $\mathbf{H} = \mathbf{I}$ (as is the case with, for example, GPV signatures [GPV08]), then we can save an additive n term in the dimension \bar{m} (and hence in the total dimension m): instead of putting an identity submatrix in $\bar{\mathbf{A}}$, we can instead use the identity submatrix from \mathbf{G} (which exists without loss of generality, since \mathbf{G} is primitive) and conceal the remainder of \mathbf{G} using either of the above methods.

All of the above ideas also translate immediately to the ring setting (see Section 4.3), using an appropriate regularity lemma (e.g., the one in [LPR10]) for a statistical instantiation, and the ring-LWE problem for a computationally secure instantiation.

5.3 LWE Inversion

Algorithm 2 below shows how to use a trapdoor to solve LWE relative to \mathbf{A} . Given a trapdoor \mathbf{R} for $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ and an LWE instance $\mathbf{b}^t = \mathbf{s}^t \mathbf{A} + \mathbf{e}^t \bmod q$ for some short error vector $\mathbf{e} \in \mathbb{Z}^m$, the algorithm recovers \mathbf{s} (and \mathbf{e}). This naturally yields an inversion algorithm for the injective trapdoor function $g_{\mathbf{A}}(\mathbf{s}, \mathbf{e}) = \mathbf{s}^t \mathbf{A} + \mathbf{e}^t \bmod q$, which is hard to invert (and whose output is pseudorandom) if LWE is hard.

Algorithm 2 Efficient algorithm $\text{Invert}^{\mathcal{O}}(\mathbf{R}, \mathbf{A}, \mathbf{b})$ for inverting the function $g_{\mathbf{A}}(\mathbf{s}, \mathbf{e})$.

Input: An oracle \mathcal{O} for inverting the function $g_{\mathbf{G}}(\hat{\mathbf{s}}, \hat{\mathbf{e}})$ when $\hat{\mathbf{e}} \in \mathbb{Z}^w$ is suitably small.

- parity-check matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$;
- \mathbf{G} -trapdoor $\mathbf{R} \in \mathbb{Z}^{\bar{m} \times kn}$ for \mathbf{A} with invertible tag $\mathbf{H} \in \mathbb{Z}_q^{n \times n}$;
- vector $\mathbf{b}^t = g_{\mathbf{A}}(\mathbf{s}, \mathbf{e}) = \mathbf{s}^t \mathbf{A} + \mathbf{e}^t$ for any $\mathbf{s} \in \mathbb{Z}_q^n$ and suitably small $\mathbf{e} \in \mathbb{Z}^m$.

Output: The vectors \mathbf{s} and \mathbf{e} .

- 1: Compute $\hat{\mathbf{b}}^t = \mathbf{b}^t \begin{bmatrix} \mathbf{R} \\ \mathbf{I} \end{bmatrix}$.
 - 2: Get $(\hat{\mathbf{s}}, \hat{\mathbf{e}}) \leftarrow \mathcal{O}(\hat{\mathbf{b}})$.
 - 3: **return** $\mathbf{s} = \mathbf{H}^{-t} \hat{\mathbf{s}}$ and $\mathbf{e} = \mathbf{b} - \mathbf{A}^t \mathbf{s}$ (interpreted as a vector in \mathbb{Z}^m with entries in $[-\frac{q}{2}, \frac{q}{2})$).
-

Theorem 5.4. Suppose that oracle \mathcal{O} in Algorithm 2 correctly inverts $g_{\mathbf{G}}(\hat{\mathbf{s}}, \hat{\mathbf{e}})$ for any error vector $\hat{\mathbf{e}} \in \mathcal{P}_{1/2}(q \cdot \mathbf{B}^{-t})$ for some \mathbf{B} . Then for any \mathbf{s} and \mathbf{e} of length $\|\mathbf{e}\| < q/(2\|\mathbf{B}\|s)$ where $s = \sqrt{s_1(\mathbf{R})^2 + 1}$, Algorithm 2 correctly inverts $g_{\mathbf{A}}(\mathbf{s}, \mathbf{e})$. Moreover, for any \mathbf{s} and random $\mathbf{e} \leftarrow D_{\mathbb{Z}^m, \alpha q}$ where $1/\alpha \geq 2\|\mathbf{B}\|s \cdot \omega(\sqrt{\log n})$, the algorithm inverts successfully with overwhelming probability over the choice of \mathbf{e} .

Note that using our constructions from Section 4, we can implement \mathcal{O} so that either $\|\mathbf{B}\| = 2$ (for q a power of 2, where $\mathbf{B} = \hat{\mathbf{S}} = 2\mathbf{I}$) or $\|\mathbf{B}\| = \sqrt{5}$ (for arbitrary q).

Proof. Let $\bar{\mathbf{R}} = [\mathbf{R}^t \mathbf{I}]$, and note that $s = s_1(\bar{\mathbf{R}})$. By the above description, the algorithm works correctly when $\bar{\mathbf{R}}\mathbf{e} \in \mathcal{P}_{1/2}(q \cdot \mathbf{B}^{-t})$; equivalently, when $(\mathbf{b}_i^t \bar{\mathbf{R}})\mathbf{e}/q \in [-\frac{1}{2}, \frac{1}{2})$ for all i . By definition of s , we have $\|\mathbf{b}_i^t \bar{\mathbf{R}}\| \leq s\|\mathbf{B}\|$. If $\|\mathbf{e}\| < q/(2\|\mathbf{B}\|s)$, then $|(\mathbf{b}_i^t \bar{\mathbf{R}})\mathbf{e}/q| < 1/2$ by Cauchy-Schwarz. Moreover, if \mathbf{e} is chosen at random from $D_{\mathbb{Z}^m, \alpha q}$, then by the fact that \mathbf{e} is 0-subgaussian (Lemma 2.8) with parameter αq , the probability that $|(\mathbf{b}_i^t \bar{\mathbf{R}})\mathbf{e}/q| \geq 1/2$ is negligible, and the second claim follows by the union bound. \square

5.4 Gaussian Sampling

Here we show how to use a trapdoor for efficient Gaussian preimage sampling for the function $f_{\mathbf{A}}$, i.e., sampling from a discrete Gaussian over a desired coset of $\Lambda^\perp(\mathbf{A})$. Our precise goal is, given a \mathbf{G} -trapdoor \mathbf{R} (with tag \mathbf{H}) for matrix \mathbf{A} and a syndrome $\mathbf{u} \in \mathbb{Z}_q^n$, to sample from the spherical discrete Gaussian $D_{\Lambda_{\mathbf{u}}^\perp(\mathbf{A}), s}$ for relatively small parameter s . As we show next, this task can be reduced, via some efficient pre- and post-processing, to sampling from any sufficiently narrow (not necessarily spherical) Gaussian over the primitive lattice $\Lambda^\perp(\mathbf{G})$.

The main ideas behind our algorithm, which is described formally in Algorithm 3, are as follows. For simplicity, suppose that \mathbf{R} has tag $\mathbf{H} = \mathbf{I}$, so $\mathbf{A} \begin{bmatrix} \mathbf{R} \\ \mathbf{I} \end{bmatrix} = \mathbf{G}$, and suppose we have a subroutine for Gaussian sampling from any desired coset of $\Lambda^\perp(\mathbf{G})$ with some small, fixed parameter $\sqrt{\Sigma_{\mathbf{G}}} \geq \eta_\epsilon(\Lambda^\perp(\mathbf{G}))$. For example, Section 4 describes algorithms for which $\sqrt{\Sigma_{\mathbf{G}}}$ is either 2 or $\sqrt{5}$. (Throughout this summary we omit the small rounding factor $r = \omega(\sqrt{\log n})$ from all Gaussian parameters.) The algorithm for sampling from a coset $\Lambda_{\mathbf{u}}^\perp(\mathbf{A})$ follows from two main observations:

1. If we sample a Gaussian \mathbf{z} with parameter $\sqrt{\Sigma_{\mathbf{G}}}$ from $\Lambda_{\mathbf{u}}^\perp(\mathbf{G})$ and produce $\mathbf{y} = \begin{bmatrix} \mathbf{R} \\ \mathbf{I} \end{bmatrix} \mathbf{z}$, then \mathbf{y} is Gaussian over the (non-full-rank) set $\begin{bmatrix} \mathbf{R} \\ \mathbf{I} \end{bmatrix} \Lambda_{\mathbf{u}}^\perp(\mathbf{G}) \subsetneq \Lambda_{\mathbf{u}}^\perp(\mathbf{A})$ with parameter $\begin{bmatrix} \mathbf{R} \\ \mathbf{I} \end{bmatrix} \sqrt{\Sigma_{\mathbf{G}}}$ (i.e., covariance $\begin{bmatrix} \mathbf{R} \\ \mathbf{I} \end{bmatrix} \Sigma_{\mathbf{G}} \begin{bmatrix} \mathbf{R}^t & \mathbf{I} \end{bmatrix}$). The (strict) inclusion holds because for any $\mathbf{y} = \begin{bmatrix} \mathbf{R} \\ \mathbf{I} \end{bmatrix} \mathbf{z}$ where $\mathbf{z} \in \Lambda_{\mathbf{u}}^\perp(\mathbf{G})$, we have

$$\mathbf{A}\mathbf{y} = (\mathbf{A} \begin{bmatrix} \mathbf{R} \\ \mathbf{I} \end{bmatrix})\mathbf{z} = \mathbf{G}\mathbf{z} = \mathbf{u}.$$

Note that $s_1(\begin{bmatrix} \mathbf{R} \\ \mathbf{I} \end{bmatrix} \cdot \sqrt{\Sigma_{\mathbf{G}}}) \leq s_1(\begin{bmatrix} \mathbf{R} \\ \mathbf{I} \end{bmatrix}) \cdot s_1(\sqrt{\Sigma_{\mathbf{G}}}) \leq \sqrt{s_1(\mathbf{R})^2 + 1} \cdot s_1(\sqrt{\Sigma_{\mathbf{G}}})$, so \mathbf{y} 's distribution is only about an $s_1(\mathbf{R})$ factor wider than that of \mathbf{z} over $\Lambda_{\mathbf{u}}^\perp(\mathbf{G})$. However, \mathbf{y} lies in a non-full-rank subset of $\Lambda_{\mathbf{u}}^\perp(\mathbf{A})$, and its distribution is ‘skewed’ (non-spherical). This leaks information about the trapdoor \mathbf{R} , so we cannot just output \mathbf{y} .

2. To sample from a *spherical* Gaussian over all of $\Lambda_{\mathbf{u}}^\perp(\mathbf{A})$, we use the ‘convolution’ technique from [Pei10] to correct for the above-described problems with the distribution of \mathbf{y} . Specifically, we first choose a Gaussian perturbation $\mathbf{p} \in \mathbb{Z}^m$ having covariance $s^2 - \begin{bmatrix} \mathbf{R} \\ \mathbf{I} \end{bmatrix} \Sigma_{\mathbf{G}} \begin{bmatrix} \mathbf{R}^t & \mathbf{I} \end{bmatrix}$, which is well-defined as long as $s \geq s_1(\begin{bmatrix} \mathbf{R} \\ \mathbf{I} \end{bmatrix} \cdot \sqrt{\Sigma_{\mathbf{G}}})$. We then sample $\mathbf{y} = \begin{bmatrix} \mathbf{R} \\ \mathbf{I} \end{bmatrix} \mathbf{z}$ as above for an adjusted syndrome $\mathbf{v} = \mathbf{u} - \mathbf{A}\mathbf{p}$, and output $\mathbf{x} = \mathbf{p} + \mathbf{y}$. Now the support of \mathbf{x} is all of $\Lambda_{\mathbf{u}}^\perp(\mathbf{A})$, and because the covariances of \mathbf{p} and \mathbf{y} are additive (subject to some mild hypotheses), the overall distribution of \mathbf{x} is spherical with Gaussian parameter s that can be as small as $s \approx s_1(\mathbf{R}) \cdot s_1(\sqrt{\Sigma_{\mathbf{G}}})$.

Quality analysis. Algorithm 3 can sample from a discrete Gaussian with parameter $s \cdot \omega(\sqrt{\log n})$ where s can be as small as $\sqrt{s_1(\mathbf{R})^2 + 1} \cdot \sqrt{s_1(\Sigma_{\mathbf{G}}) + 2}$. We stress that this is only very slightly larger — a factor of at most $\sqrt{6/4} \leq 1.23$ — than the bound $(s_1(\mathbf{R}) + 1) \cdot \|\tilde{\mathbf{S}}\|$ from Lemma 5.3 on the largest Gram-Schmidt norm of a lattice basis derived from the trapdoor \mathbf{R} . (Recall that our constructions from Section 4 give $s_1(\Sigma_{\mathbf{G}}) = \|\tilde{\mathbf{S}}\|^2 = 4$ or 5 .) In the iterative “randomized nearest-plane” sampling algorithm of [Kle00, GPV08], the Gaussian parameter s is lower-bounded by the largest Gram-Schmidt norm of the orthogonalized input basis (times the same $\omega(\sqrt{\log n})$ factor used in our algorithm). Therefore, the efficiency

and parallelism of Algorithm 3 comes at almost no cost in quality versus slower, iterative algorithms that use high-precision arithmetic. (It seems very likely that the corresponding small loss in security can easily be mitigated with slightly larger parameters, while still yielding a significant net gain in performance.)

Runtime analysis. We now analyze the computational cost of Algorithm 3, with a focus on optimizing the online runtime and parallelism (sometimes at the expense of the offline phase, which we do not attempt to optimize).

The offline phase is dominated by sampling from $D_{\mathbb{Z}^m, r\sqrt{\Sigma}}$ for some fixed (typically non-spherical) covariance matrix $\Sigma > \mathbf{I}$. By [Pei10, Theorem 3.1], this can be accomplished (up to any desired statistical distance) simply by sampling a continuous Gaussian $D_{r\sqrt{\Sigma-\mathbf{I}}}$ with sufficient precision, then independently randomized-rounding each entry of the sampled vector to \mathbb{Z} using Gaussian parameter $r \geq \eta_\epsilon(\mathbb{Z})$.

Naively, the online work is dominated by the computation of $\mathbf{H}^{-1}(\mathbf{u} - \bar{\mathbf{w}})$ and $\mathbf{R}\mathbf{z}$ (plus the call to $\mathcal{O}(\mathbf{v})$, which as described in Section 4 requires only $O(\log^c n)$ work, or one table lookup, by each of n processors in parallel). In general, the first computation takes $O(n^2)$ scalar multiplications and additions in \mathbb{Z}_q , while the latter takes $O(\bar{m} \cdot w)$, which is typically $\Theta(n^2 \log^2 q)$. (Obviously, both computations are perfectly parallelizable.) However, the special form of \mathbf{z} , and often of \mathbf{H} , allow for some further asymptotic and practical optimizations: since \mathbf{z} is typically produced by concatenating n independent dimension- k subvectors that are sampled offline, we can precompute much of $\mathbf{R}\mathbf{z}$ by pre-multiplying each subvector by each of the n blocks of k columns in \mathbf{R} . This reduces the online computation of $\mathbf{R}\mathbf{z}$ to the summation of n dimension- \bar{m} vectors, or $O(n^2 \log q)$ scalar additions (and no multiplications) in \mathbb{Z}_q . As for multiplication by \mathbf{H}^{-1} , in some applications (like GPV signatures) \mathbf{H} is always the identity \mathbf{I} , in which case multiplication is unnecessary; in all other applications we know of, \mathbf{H} actually represents multiplication in a certain extension field/ring of \mathbb{Z}_q , which can be computed in $O(n \log n)$ scalar operations and depth $O(\log n)$. In conclusion, the asymptotic cost of the online phase is still dominated by computing $\mathbf{R}\mathbf{z}$, which takes $\tilde{O}(n^2)$ work, but the hidden constants are small and many practical speedups are possible.

Theorem 5.5. *Algorithm 3 is correct.*

To prove the theorem we need the following fact about products of Gaussian functions.

Fact 5.6 (Product of degenerate Gaussians). *Let $\Sigma_1, \Sigma_2 \in \mathbb{R}^{m \times m}$ be symmetric positive semidefinite matrices, let $V_i = \text{span}(\Sigma_i)$ for $i = 1, 2$ and $V_3 = V_1 \cap V_2$, let $\mathbf{P} = \mathbf{P}^t \in \mathbb{R}^{m \times m}$ be the symmetric matrix that projects orthogonally onto V_3 , and let $\mathbf{c}_1, \mathbf{c}_2 \in \mathbb{R}^m$ be arbitrary. Supposing it exists, let \mathbf{v} be the unique point in $(V_1 + \mathbf{c}_1) \cap (V_2 + \mathbf{c}_2) \cap V_3^\perp$. Then*

$$\rho_{\sqrt{\Sigma_1}}(\mathbf{x} - \mathbf{c}_1) \cdot \rho_{\sqrt{\Sigma_2}}(\mathbf{x} - \mathbf{c}_2) = \rho_{\sqrt{\Sigma_1 + \Sigma_2}}(\mathbf{c}_1 - \mathbf{c}_2) \cdot \rho_{\sqrt{\Sigma_3}}(\mathbf{x} - \mathbf{c}_3),$$

where Σ_3 and $\mathbf{c}_3 \in \mathbf{v} + V_3$ are such that

$$\begin{aligned} \Sigma_3^+ &= \mathbf{P}(\Sigma_1^+ + \Sigma_2^+)\mathbf{P} \\ \Sigma_3^+(\mathbf{c}_3 - \mathbf{v}) &= \Sigma_1^+(\mathbf{c}_1 - \mathbf{v}) + \Sigma_2^+(\mathbf{c}_2 - \mathbf{v}). \end{aligned}$$

Proof of Theorem 5.5. We adopt the notation from the algorithm, let $V = \text{span}(\begin{bmatrix} \mathbf{R} \\ \mathbf{I} \end{bmatrix}) \subset \mathbb{R}^m$, let \mathbf{P} be the matrix that projects orthogonally onto V , and define the lattice $\Lambda = \mathbb{Z}^m \cap V = \mathcal{L}(\begin{bmatrix} \mathbf{R} \\ \mathbf{I} \end{bmatrix})$, which spans V . We analyze the output distribution of SampleD. Clearly, it always outputs an element of $\Lambda_{\mathbf{u}}^\perp(\mathbf{A})$, so let $\bar{\mathbf{x}} \in \Lambda_{\mathbf{u}}^\perp(\mathbf{A})$ be arbitrary. Now SampleD outputs $\bar{\mathbf{x}}$ exactly when it chooses in Step 1 some $\bar{\mathbf{p}} \in V + \bar{\mathbf{x}}$, followed in

Algorithm 3 Efficient algorithm $\text{SampleD}^{\mathcal{O}}(\mathbf{R}, \bar{\mathbf{A}}, \mathbf{H}, \mathbf{u}, s)$ for sampling a discrete Gaussian over $\Lambda_{\mathbf{u}}^{\perp}(\mathbf{A})$.

Input: An oracle $\mathcal{O}(\mathbf{v})$ for Gaussian sampling over a desired coset $\Lambda_{\mathbf{v}}^{\perp}(\mathbf{G})$ with fixed parameter $r\sqrt{\Sigma_{\mathbf{G}}} \geq \eta_{\epsilon}(\Lambda^{\perp}(\mathbf{G}))$, for some $\Sigma_{\mathbf{G}} \geq 2$ and $\epsilon \leq 1/2$.

Offline phase:

- partial parity-check matrix $\bar{\mathbf{A}} \in \mathbb{Z}_q^{n \times \bar{m}}$;
- trapdoor matrix $\mathbf{R} \in \mathbb{Z}^{\bar{m} \times w}$;
- positive definite $\Sigma \geq \begin{bmatrix} \mathbf{R} \\ \mathbf{I} \end{bmatrix} (2 + \Sigma_{\mathbf{G}}) \begin{bmatrix} \mathbf{R}^t & \mathbf{I} \end{bmatrix}$, e.g., any $\Sigma = s^2 \geq (s_1(\mathbf{R})^2 + 1)(s_1(\Sigma_{\mathbf{G}}) + 2)$.

Online phase:

- invertible tag $\mathbf{H} \in \mathbb{Z}_q^{n \times n}$ defining $\mathbf{A} = [\bar{\mathbf{A}} \mid \mathbf{H}\mathbf{G} - \bar{\mathbf{A}}\mathbf{R}] \in \mathbb{Z}_q^{n \times m}$, for $m = \bar{m} + w$ (\mathbf{H} may instead be provided in the offline phase, if it is known then);
- syndrome $\mathbf{u} \in \mathbb{Z}_q^n$.

Output: A vector \mathbf{x} drawn from a distribution within $O(\epsilon)$ statistical distance of $D_{\Lambda_{\mathbf{u}}^{\perp}(\mathbf{A}), r\sqrt{\Sigma}}$.

Offline phase:

- 1: Choose a fresh perturbation $\mathbf{p} \leftarrow D_{\mathbb{Z}^m, r\sqrt{\Sigma_{\mathbf{p}}}}$, where $\Sigma_{\mathbf{p}} = \Sigma - \begin{bmatrix} \mathbf{R} \\ \mathbf{I} \end{bmatrix} \Sigma_{\mathbf{G}} \begin{bmatrix} \mathbf{R}^t & \mathbf{I} \end{bmatrix} \geq 2 \begin{bmatrix} \mathbf{R} \\ \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{R}^t & \mathbf{I} \end{bmatrix}$.
- 2: Let $\mathbf{p} = \begin{bmatrix} \mathbf{p}_1 \\ \mathbf{p}_2 \end{bmatrix}$ for $\mathbf{p}_1 \in \mathbb{Z}^{\bar{m}}$, $\mathbf{p}_2 \in \mathbb{Z}^w$, and compute $\bar{\mathbf{w}} = \bar{\mathbf{A}}(\mathbf{p}_1 - \mathbf{R}\mathbf{p}_2) \in \mathbb{Z}_q^n$ and $\mathbf{w} = \mathbf{G}\mathbf{p}_2 \in \mathbb{Z}_q^n$.

Online phase:

- 3: Let $\mathbf{v} \leftarrow \mathbf{H}^{-1}(\mathbf{u} - \bar{\mathbf{w}}) - \mathbf{w} = \mathbf{H}^{-1}(\mathbf{u} - \mathbf{A}\mathbf{p}) \in \mathbb{Z}_q^n$, and choose $\mathbf{z} \leftarrow D_{\Lambda_{\mathbf{v}}^{\perp}(\mathbf{G}), r\sqrt{\Sigma_{\mathbf{G}}}}$ by calling $\mathcal{O}(\mathbf{v})$.
 - 4: **return** $\mathbf{x} \leftarrow \mathbf{p} + \begin{bmatrix} \mathbf{R} \\ \mathbf{I} \end{bmatrix} \mathbf{z}$.
-

Step 3 by the unique $\bar{\mathbf{z}} \in \Lambda_{\mathbf{v}}^{\perp}(\mathbf{G})$ such that $\bar{\mathbf{x}} - \bar{\mathbf{p}} = \begin{bmatrix} \mathbf{R} \\ \mathbf{I} \end{bmatrix} \bar{\mathbf{z}}$. It is easy to check that $\rho_{\sqrt{\Sigma_{\mathbf{G}}}}(\bar{\mathbf{z}}) = \rho_{\sqrt{\Sigma_{\mathbf{y}}}}(\bar{\mathbf{x}} - \bar{\mathbf{p}})$, where

$$\Sigma_{\mathbf{y}} = \begin{bmatrix} \mathbf{R} \\ \mathbf{I} \end{bmatrix} \Sigma_{\mathbf{G}} \begin{bmatrix} \mathbf{R}^t & \mathbf{I} \end{bmatrix} \geq 2 \begin{bmatrix} \mathbf{R} \\ \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{R}^t & \mathbf{I} \end{bmatrix}$$

is the covariance matrix with $\text{span}(\Sigma_{\mathbf{y}}) = V$. Note that $\Sigma_{\mathbf{p}} + \Sigma_{\mathbf{y}} = \Sigma$ by definition of $\Sigma_{\mathbf{p}}$, and that $\text{span}(\Sigma_{\mathbf{p}}) = \mathbb{R}^m$ because $\Sigma_{\mathbf{p}} > \mathbf{0}$. Therefore, we have (where C denotes a normalizing constant that may vary from line to line, but does not depend on $\bar{\mathbf{x}}$):

$$\begin{aligned}
p_{\bar{\mathbf{x}}} &= \Pr[\text{SampleD outputs } \bar{\mathbf{x}}] \\
&= \sum_{\bar{\mathbf{p}} \in \mathbb{Z}^m \cap (V + \bar{\mathbf{x}})} D_{\mathbb{Z}^m, r\sqrt{\Sigma_{\mathbf{p}}}}(\bar{\mathbf{p}}) \cdot D_{\Lambda_{\mathbf{v}}^{\perp}(\mathbf{G}), r\sqrt{\Sigma_{\mathbf{y}}}}(\bar{\mathbf{z}}) && \text{(def. of SampleD)} \\
&= C \sum_{\bar{\mathbf{p}}} \rho_{r\sqrt{\Sigma_{\mathbf{p}}}}(\bar{\mathbf{p}}) \cdot \rho_{r\sqrt{\Sigma_{\mathbf{y}}}}(\bar{\mathbf{p}} - \bar{\mathbf{x}}) / \rho_{r\sqrt{\Sigma_{\mathbf{G}}}}(\Lambda_{\mathbf{v}}^{\perp}(\mathbf{G})) && \text{(def. of } D) \\
&= C \cdot \rho_{r\sqrt{\Sigma}}(\bar{\mathbf{x}}) \cdot \sum_{\bar{\mathbf{p}}} \rho_{r\sqrt{\Sigma_3}}(\bar{\mathbf{p}} - \mathbf{c}_3) / \rho_{r\sqrt{\Sigma_{\mathbf{G}}}}(\Lambda_{\mathbf{v}}^{\perp}(\mathbf{G})) && \text{(Fact 5.6)} \\
&\in C[1, \frac{1+\epsilon}{1-\epsilon}] \cdot \rho_{r\sqrt{\Sigma}}(\bar{\mathbf{x}}) \cdot \sum_{\bar{\mathbf{p}}} \rho_{r\sqrt{\Sigma_3}}(\bar{\mathbf{p}} - \mathbf{c}_3) && \text{(Lemma 2.5 and } r\sqrt{\Sigma_{\mathbf{G}}} \geq \eta_{\epsilon}(\Lambda^{\perp}(\mathbf{G}))) \\
&= C[1, \frac{1+\epsilon}{1-\epsilon}] \cdot \rho_{r\sqrt{\Sigma}}(\bar{\mathbf{x}}) \cdot \rho_{r\sqrt{\Sigma_3}}(\mathbb{Z}^m \cap (V + \bar{\mathbf{x}}) - \mathbf{c}_3), && (5.1)
\end{aligned}$$

where $\Sigma_3^+ = \mathbf{P}(\Sigma_{\mathbf{p}}^+ + \Sigma_{\mathbf{y}}^+)\mathbf{P}$ and $\mathbf{c}_3 \in \mathbf{v} + V = \bar{\mathbf{x}} + V$, because the component of $\bar{\mathbf{x}}$ orthogonal to V is the unique point $\mathbf{v} \in (V + \bar{\mathbf{x}}) \cap V^{\perp}$. Therefore,

$$\mathbb{Z}^m \cap (V + \bar{\mathbf{x}}) - \mathbf{c}_3 = (\mathbb{Z}^m \cap V) + (\bar{\mathbf{x}} - \mathbf{c}_3) \subset V$$

is a coset of the lattice $\Lambda = \mathcal{L}(\begin{bmatrix} \mathbf{R} \\ \mathbf{I} \end{bmatrix})$. It remains to show that $r\sqrt{\Sigma_3} \geq \eta_\epsilon(\Lambda)$, so that the rightmost term in (5.1) above is essentially a constant (up to some factor in $[\frac{1-\epsilon}{1+\epsilon}, 1]$) independent of $\bar{\mathbf{x}}$, by Lemma 2.5. Then we can conclude that $p_{\bar{\mathbf{x}}} \in [\frac{1-\epsilon}{1+\epsilon}, \frac{1+\epsilon}{1-\epsilon}] \cdot \rho_{r\sqrt{\Sigma}(\bar{\mathbf{x}})$, from which the theorem follows.

To show that $r\sqrt{\Sigma_3} \geq \eta_\epsilon(\Lambda)$, note that since $\Lambda^* \subset V$, for any covariance Π we have $\rho_{\mathbf{P}\sqrt{\Pi}}(\Lambda^*) = \rho_{\sqrt{\Pi}}(\Lambda^*)$, and so $\mathbf{P}\sqrt{\Pi} \geq \eta_\epsilon(\Lambda)$ if and only if $\sqrt{\Pi} \geq \eta_\epsilon(\Lambda)$. Now because both $\Sigma_{\mathbf{p}}, \Sigma_{\mathbf{y}} \geq 2\begin{bmatrix} \mathbf{R} \\ \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{R}^t & \mathbf{I} \end{bmatrix}$, we have

$$\Sigma_{\mathbf{p}}^+ + \Sigma_{\mathbf{y}}^+ \leq (\begin{bmatrix} \mathbf{R} \\ \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{R}^t & \mathbf{I} \end{bmatrix})^+.$$

Because $r\begin{bmatrix} \mathbf{R} \\ \mathbf{I} \end{bmatrix} \geq \eta_\epsilon(\Lambda)$ for $\epsilon = \text{negl}(n)$ by Lemma 2.3, we have $r\sqrt{\Sigma_3} = r\sqrt{(\Sigma_{\mathbf{p}}^+ + \Sigma_{\mathbf{y}}^+)^+} \geq \eta_\epsilon(\Lambda)$, as desired. \square

5.5 Trapdoor Delegation

Here we describe very simple and efficient mechanism for securely delegating a trapdoor for $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ to a trapdoor for an extension $\mathbf{A}' \in \mathbb{Z}_q^{n \times m'}$ of \mathbf{A} . Our method has several advantages over the previous basis delegation algorithm of [CHKP10]: first and most importantly, the size of the delegated trapdoor grows only linearly with the dimension m' of $\Lambda^\perp(\mathbf{A}')$, rather than quadratically. Second, the algorithm is much more efficient, because it does not require testing linear independence of Gaussian samples, nor computing the expensive ToBasis and Hermite normal form operations. Third, the resulting trapdoor \mathbf{R} has a ‘nice’ Gaussian distribution that is easy to analyze and may be useful in applications. We do note that while the delegation algorithm from [CHKP10] works for *any* extension \mathbf{A}' of \mathbf{A} (including \mathbf{A} itself), ours requires $m' \geq m + w$. Fortunately, this is frequently the case in applications such as HIBE and others that use delegation.

Algorithm 4 Efficient algorithm $\text{DelTrap}^\mathcal{O}(\mathbf{A}' = [\mathbf{A} \mid \mathbf{A}_1], \mathbf{H}', s')$ for delegating a trapdoor.

Input: an oracle \mathcal{O} for discrete Gaussian sampling over cosets of $\Lambda = \Lambda^\perp(\mathbf{A})$ with parameter $s' \geq \eta_\epsilon(\Lambda)$.

- parity-check matrix $\mathbf{A}' = [\mathbf{A} \mid \mathbf{A}_1] \in \mathbb{Z}_q^{n \times m} \times \mathbb{Z}_q^{n \times w}$;
- invertible matrix $\mathbf{H}' \in \mathbb{Z}_q^{n \times n}$;

Output: a trapdoor $\mathbf{R}' \in \mathbb{Z}^{m \times w}$ for \mathbf{A}' with tag $\mathbf{H} \in \mathbb{Z}_q^{n \times n}$.

- 1: Using \mathcal{O} , sample each column of \mathbf{R}' independently from a discrete Gaussian with parameter s' over the appropriate coset of $\Lambda^\perp(\mathbf{A})$, so that $\mathbf{A}\mathbf{R}' = \mathbf{H}'\mathbf{G} - \mathbf{A}_1$.
-

Usually, the oracle \mathcal{O} needed by Algorithm 4 would be implemented (up to $\text{negl}(n)$ statistical distance) by Algorithm 3 above, using a trapdoor \mathbf{R} for \mathbf{A} where $s_1(\mathbf{R})$ is sufficiently small relative to s' . The following is immediate from Lemma 2.9 and the fact that the columns of \mathbf{R}' are independent and $\text{negl}(n)$ -subgaussian. A relatively tight bound on the hidden constant factor can also be derived from Lemma 2.9.

Lemma 5.7. *For any valid inputs \mathbf{A}' and \mathbf{H}' , Algorithm 4 outputs a trapdoor \mathbf{R}' for \mathbf{A}' with tag \mathbf{H}' , whose distribution is the same for any valid implementation of \mathcal{O} , and $s_1(\mathbf{R}') \leq s' \cdot O(\sqrt{m} + \sqrt{w})$ except with negligible probability.*

6 Applications

The main applications of “strong” trapdoors have included digital signature schemes in both the random-oracle and standard models, encryption secure under chosen-ciphertext attack (CCA), and (hierarchical) identity-based encryption. Here we focus on signature schemes and CCA-secure encryption, where our techniques lead to significant new improvements (beyond what is obtained by plugging in our trapdoor generator as a “black box”). Where appropriate, we also briefly mention the improvements that are possible in the remaining applications.

6.1 Algebraic Background

In our applications we need a special collection of elements from a certain ring \mathcal{R} , which induce invertible matrices $\mathbf{H} \in \mathbb{Z}_q^{n \times n}$ as required by our trapdoor construction. We construct such a ring using ideas from the literature on secret sharing over groups and modules, e.g., [DF94, Feh98]. Define the ring $\mathcal{R} = \mathbb{Z}_q[x]/(f(x))$ for some monic degree- n polynomial $f(x) = x^n + f_{n-1}x^{n-1} + \dots + f_0 \in \mathbb{Z}[x]$ that is irreducible modulo every prime p dividing q . (Such an $f(x)$ can be constructed by finding monic irreducible degree- n polynomials in $\mathbb{Z}_p[x]$ for each prime p dividing q , and using the Chinese remainder theorem on their coefficients to get $f(x)$.) Recall that \mathcal{R} is a free \mathbb{Z}_q -module of rank n , i.e., the elements of \mathcal{R} can be represented as vectors in \mathbb{Z}_q^n relative to the standard basis of monomials $1, x, \dots, x^{n-1}$. Multiplication by any fixed element of \mathcal{R} then acts as a linear transformation on \mathbb{Z}_q^n according to the rule $x \cdot (a_0, \dots, a_{n-1})^t = (0, a_0, \dots, a_{n-2})^t - a_{n-1}(f_0, f_1, \dots, f_{n-1})^t$, and so can be represented by an (efficiently computable) matrix in $\mathbb{Z}_q^{n \times n}$ relative to the standard basis. In other words, there is an injective ring homomorphism $h: \mathcal{R} \rightarrow \mathbb{Z}_q^{n \times n}$ that maps any $a \in \mathcal{R}$ to the matrix $\mathbf{H} = h(a)$ representing multiplication by a . In particular, \mathbf{H} is invertible if and only if $a \in \mathcal{R}^*$, the set of units in \mathcal{R} . By the Chinese remainder theorem, and because $\mathbb{Z}_p[x]/(f(x))$ is a field by construction of $f(x)$, an element $a \in \mathcal{R}$ is a unit exactly when it is nonzero (as a polynomial residue) modulo every prime p dividing q . We use this fact quite essentially in the constructions that follow.

6.2 Signature Schemes

6.2.1 Definitions

A signature scheme SIG for a message space \mathcal{M} (which may depend on the security parameter n) is a tuple of PPT algorithms as follows:

- $\text{Gen}(1^n)$ outputs a verification key vk and a signing key sk .
- $\text{Sign}(sk, \mu)$, given a signing key sk and a message $\mu \in \mathcal{M}$, outputs a signature $\sigma \in \{0, 1\}^*$.
- $\text{Ver}(vk, \mu, \sigma)$, given a verification key vk , a message μ , and a signature σ , either accepts or rejects.

The correctness requirement is: for any $\mu \in \mathcal{M}$, generate $(vk, sk) \leftarrow \text{Gen}(1^n)$ and $\sigma \leftarrow \text{Sign}(sk, \mu)$. Then $\text{Ver}(vk, \mu, \sigma)$ should accept with overwhelming probability (over all the randomness in the experiment).

We recall two standard notions of security for signatures. An intermediate notion is strong unforgeability under *static* chosen-message attack, or su-scma security, is defined as follows: first, the forger \mathcal{F} outputs a list of *distinct* query messages $\mu^{(1)}, \dots, \mu^{(Q)}$ for some Q . (The distinctness condition simplifies our construction, and does not affect the notion’s usefulness.) Next, we generate $(vk, sk) \leftarrow \text{Gen}(1^n)$ and $\sigma^{(i)} \leftarrow \text{Sign}(sk, \mu^{(i)})$ for each $i \in [Q]$, then give vk and each $\sigma^{(i)}$ to \mathcal{F} . Finally, \mathcal{F} outputs an attempted forgery (μ^*, σ^*) . The forger’s advantage $\text{Adv}_{\text{SIG}}^{\text{su-scma}}(\mathcal{F})$ is the probability that $\text{Ver}(vk, \mu^*, \sigma^*)$ accepts and $(\mu^*, \sigma^*) \neq (\mu^{(i)}, \sigma^{(i)})$ for all $i \in [Q]$, taken over all the randomness of the experiment. The

scheme is su-scma-secure if $\text{Adv}_{\text{SIG}}^{\text{su-scma}}(\mathcal{F}) = \text{negl}(n)$ for every nonuniform probabilistic polynomial-time algorithm \mathcal{F} .

Another notion, called strong existential unforgeability under *adaptive* chosen-message attack, or su-acma security, is defined similarly, except that \mathcal{F} is first given vk and may adaptively choose the messages $\mu^{(i)}$ to be signed, which need not be distinct.

Using a family of *chameleon* hash functions, there is a generic transformation from eu-scma- to eu-acma-security; see, e.g., [KR00]. Furthermore, the transformation results in an *offline/online* scheme in which the Sign algorithm can be precomputed before the message to be signed is known; see [ST01]. The basic idea is that the signer chameleon hashes the true message, then signs the hash value using the eu-scma-secure scheme (and includes the randomness used in the chameleon hash with the final signature). A suitable type of chameleon hash function has been constructed under a weak hardness-of-SIS assumption; see [CHKP10].

6.2.2 Standard Model Scheme

Here we give a signature scheme that is statically secure in the standard model. The scheme itself is essentially identical (up to the improved and generalized parameters) to the one of [Boy10], which is a lattice analogue of the pairing-based signature of [Wat05]. We give a new proof with an improved security reduction that relies on a weaker assumption. The proof uses a variant of the “prefix technique” [HW09] also used in [CHKP10].

Our scheme involves a number of parameters. For simplicity, we give some exemplary asymptotic bounds here. (Other slight trade-offs among the parameters are possible, and more precise values can be obtained using the more exact bounds from earlier in the paper and the material below.) In what follows, $\omega(\sqrt{\log n})$ represents a fixed function that asymptotically grows faster than $\sqrt{\log n}$.

- $\mathbf{G} \in \mathbb{Z}_q^{n \times nk}$ is a gadget matrix for large enough $q = \text{poly}(n)$ and $k = \lceil \log q \rceil = O(\log n)$, with the ability to sample from cosets of $\Lambda^\perp(\mathbf{G})$ with Gaussian parameter $O(1) \cdot \omega(\sqrt{\log n}) \geq \eta_\epsilon(\Lambda^\perp(\mathbf{G}))$. (See for example the constructions from Section 4.)
- $\bar{m} = O(nk)$ and $\mathcal{D} = D_{\mathbb{Z}, \omega(\sqrt{\log n})}^{\bar{m} \times nk}$ so that $(\bar{\mathbf{A}}, \bar{\mathbf{A}}\mathbf{R})$ is $\text{negl}(n)$ -far from uniform for $\bar{\mathbf{A}} \leftarrow \mathbb{Z}_q^{n \times \bar{m}}$ and $\mathbf{R} \leftarrow \mathcal{D}$, and $m = \bar{m} + 2nk$ is the total dimension of the signatures.
- ℓ is a suitable message length (see below), and $s = O(\sqrt{\ell nk}) \cdot \omega(\sqrt{\log n})^2$ is a sufficiently large Gaussian parameter.

The legal values of ℓ are influenced by the choice of q and n . Our security proof requires a special collection of units in the ring $\mathcal{R} = \mathbb{Z}_q[x]/(f(x))$ as constructed in Section 6.1 above. We need a sequence of ℓ units $u_1, \dots, u_\ell \in \mathcal{R}^*$, not necessarily distinct, such that any nontrivial subset-sum is also a unit, i.e., for any nonempty $S \subseteq [\ell]$, $\sum_{i \in S} u_i \in \mathcal{R}^*$. By the characterization of units in \mathcal{R} described in Section 6.1, letting p be the smallest prime dividing q , we can allow any $\ell \leq (p-1) \cdot n$ by taking $p-1$ copies of each of the monomials $x^i \in \mathcal{R}^*$ for $i = 0, \dots, n-1$.

The signature scheme has message space $\{0, 1\}^\ell$, and is defined as follows.

- **Gen**(1^n): choose $\bar{\mathbf{A}} \leftarrow \mathbb{Z}_q^{n \times \bar{m}}$, choose $\mathbf{R} \in \mathbb{Z}^{\bar{m} \times nk}$ from distribution \mathcal{D} , and let $\mathbf{A} = [\bar{\mathbf{A}} \mid \mathbf{G} - \bar{\mathbf{A}}\mathbf{R}]$. For $i = 0, 1, \dots, \ell$, choose $\mathbf{A}_i \leftarrow \mathbb{Z}_q^{n \times nk}$. Also choose a syndrome $\mathbf{u} \leftarrow \mathbb{Z}_q^n$. The public verification key is $vk = (\mathbf{A}, \mathbf{A}_0, \dots, \mathbf{A}_\ell, \mathbf{u})$. The secret signing key is $sk = \mathbf{R}$.
- **Sign**($sk, \mu \in \{0, 1\}^\ell$): let $\mathbf{A}_\mu = [\mathbf{A} \mid \mathbf{A}_0 + \sum_{i \in [\ell]} \mu_i \mathbf{A}_i] \in \mathbb{Z}_q^{n \times m}$, where $\mu_i \in \{0, 1\}$ is the i th bit of μ , interpreted as an integer. Output $\mathbf{v} \in \mathbb{Z}^m$ sampled from $D_{\Lambda_{\mathbf{u}}^\perp(\mathbf{A}_\mu), s}$, using SampleD with trapdoor \mathbf{R} for \mathbf{A} (which is also a trapdoor for its extension \mathbf{A}_μ).

- $\text{Ver}(vk, \mu, \mathbf{v})$: let \mathbf{A}_μ be as above. Accept if $\|\mathbf{v}\| \leq s \cdot \sqrt{m}$ and $\mathbf{A}_\mu \cdot \mathbf{v} = \mathbf{u}$; otherwise, reject.

Notice that the signing process takes $O(\ell n^2 k)$ scalar operations (to add up the \mathbf{A}_i s), but after transforming the scheme to a fully secure one using chameleon hashing, these computations can be performed offline before the message is known.

Theorem 6.1. *There exists a PPT oracle algorithm (a reduction) \mathcal{S} attacking the $\text{SIS}_{q,\beta}$ problem for large enough $\beta = O(\ell(nk)^{3/2}) \cdot \omega(\sqrt{\log n})^3$ such that, for any adversary \mathcal{F} mounting an su-scma attack on SIG and making at most Q queries,*

$$\text{Adv}_{\text{SIS}_{q,\beta}}(\mathcal{S}^\mathcal{F}) \geq \text{Adv}_{\text{SIG}}^{\text{su-scma}}(\mathcal{F}) / (2(\ell - 1)Q + 2) - \text{negl}(n).$$

Proof. Let \mathcal{F} be an adversary mounting an su-scma attack on SIG, having advantage $\delta = \text{Adv}_{\text{SIG}}^{\text{su-scma}}(\mathcal{F})$. We construct a reduction \mathcal{S} attacking $\text{SIS}_{q,\beta}$. The reduction \mathcal{S} takes as input $\bar{m} + nk + 1$ uniformly random and independent samples from \mathbb{Z}_q^n , parsing them as a matrix $\mathbf{A} = [\bar{\mathbf{A}} \mid \mathbf{B}] \in \mathbb{Z}_q^{n \times (\bar{m} + nk)}$ and syndrome $\mathbf{u}' \in \mathbb{Z}_q^n$. It will use \mathcal{F} either to find some $\mathbf{z} \in \mathbb{Z}^m$ of length $\|\mathbf{z}\| \leq \beta - 1$ such that $\mathbf{A}\mathbf{z} = \mathbf{u}'$ (from which it follows that $[\mathbf{A} \mid \mathbf{u}'] \cdot \mathbf{z}' = \mathbf{0}$, where $\mathbf{z}' = [\mathbf{z}; -1]$ is nonzero and of length at most β), or a nonzero $\mathbf{z} \in \mathbb{Z}^m$ such that $\mathbf{A}\mathbf{z} = \mathbf{0}$ (from which it follows that $[\mathbf{A} \mid \mathbf{u}'] \cdot [\mathbf{z}; 0] = \mathbf{0}$).

We distinguish between two types of forger \mathcal{F} : one that produces a forgery on an unqueried message (a violation of standard existential unforgeability), and one that produces a new signature on a queried message (a violation of strong unforgeability). Clearly any \mathcal{F} with advantage δ has probability at least $\delta/2$ of succeeding in at least one of these two tasks.

First we consider \mathcal{F} that forges on an unqueried message (with probability at least $\delta/2$). Our reduction \mathcal{S} simulates the static chosen-message attack to \mathcal{F} as follows:

- Invoke \mathcal{F} to receive up to Q messages $\mu^{(1)}, \mu^{(2)}, \dots \in \{0, 1\}^\ell$. Compute the set P of all strings $p \in \{0, 1\}^{\leq \ell}$ having the property that p is a shortest string for which no $\mu^{(j)}$ has p as a prefix. Equivalently, P represents the set of maximal subtrees of $\{0, 1\}^{\leq \ell}$ (viewed as a tree) that do not contain any of the queried messages. The set P has size at most $(\ell - 1) \cdot Q + 1$, and may be computed efficiently. (See, e.g., [CHKP10] for a precise description of an algorithm.) Choose some p from P uniformly at random, letting $t = |p| \leq \ell$.
- Construct a verification key $vk = (\mathbf{A}, \mathbf{A}_0, \dots, \mathbf{A}_\ell, \mathbf{u} = \mathbf{u}')$: for $i = 0, \dots, \ell$, choose $\mathbf{R}_i \leftarrow \mathcal{D}$, and let

$$\mathbf{A}_i = \mathbf{H}_i \mathbf{G} - \bar{\mathbf{A}} \mathbf{R}_i, \text{ where } \mathbf{H}_i = \begin{cases} h(0) = \mathbf{0} & i > t \\ (-1)^{p_i} \cdot h(u_i) & i \in [t] \\ -\sum_{j \in [t]} p_j \cdot \mathbf{H}_j & i = 0 \end{cases}$$

(Recall that $u_1, \dots, u_\ell \in \mathcal{R} = \mathbb{Z}_q[x]/(f(x))$ are units whose nontrivial subset-sums are also units.)

Note that by hypothesis on \bar{m} and \mathcal{D} , for any choice of p the key vk is only $\text{negl}(n)$ -far from uniform in statistical distance. Note also that by our choice of the \mathbf{H}_i , for any message $\mu \in \{0, 1\}^\ell$ having p as a prefix, we have $\mathbf{H}_0 + \sum_{i \in [\ell]} \mu_i \mathbf{H}_i = \mathbf{0}$. Whereas for any $\mu \in \{0, 1\}^\ell$ having $p' \neq p$ as its t -bit prefix, we have

$$\mathbf{H}_0 + \sum_{i \in [\ell]} \mu_i \mathbf{H}_i = \sum_{i \in [t]} (p'_i - p_i) \cdot \mathbf{H}_i = \sum_{i \in [t], p'_i \neq p_i} (-1)^{p_i} \cdot \mathbf{H}_i = h\left(\sum_{i \in [t], p'_i \neq p_i} u_i\right),$$

which is invertible by hypothesis on the u_i s. Finally, observe that with overwhelming probability over any fixed choice of vk and the \mathbf{H}_i , each column of each \mathbf{R}_i is still independently distributed as

a discrete Gaussian with parameter $\omega(\sqrt{\log n}) \geq \eta_\epsilon(\bar{\mathbf{A}})$ over some fixed coset of $\Lambda^\perp(\bar{\mathbf{A}})$, for some negligible $\epsilon = \epsilon(n)$.

- Generate signatures for the queried messages: for each message $\mu = \mu^{(i)}$, compute

$$\mathbf{A}_\mu = [\mathbf{A} \mid \mathbf{A}_0 + \sum_{i \in [\ell]} \mu_i \mathbf{A}_i] = [\bar{\mathbf{A}} \mid \mathbf{B} \mid \mathbf{H}\mathbf{G} - \bar{\mathbf{A}}(\mathbf{R}_0 + \sum_{i \in [\ell]} \mu_i \mathbf{R}_i)],$$

where \mathbf{H} is invertible because the t -bit prefix of μ is not p . Therefore, $\mathbf{R} = (\mathbf{R}_0 + \sum_{i \in [\ell]} \mu_i \mathbf{R}_i)$ is a trapdoor for \mathbf{A}_μ . By the conditional distribution on the \mathbf{R}_i s, concatenation of subgaussian random variables, and Lemma 2.9, we have

$$s_1(\mathbf{R}) = \sqrt{\ell+1} \cdot O(\sqrt{m} + \sqrt{nk}) \cdot \omega(\sqrt{\log n}) = O(\sqrt{\ell nk}) \cdot \omega(\sqrt{\log n})$$

with overwhelming probability. Since $s = O(\sqrt{\ell nk}) \cdot \omega(\sqrt{\log n})^2$ is sufficiently large, we can generate a properly distributed signature $\mathbf{v}_\mu \leftarrow D_{\Lambda_{\mathbf{u}}^\perp(\mathbf{A}_\mu), s}$ using SampleD with trapdoor \mathbf{R} .

Next, \mathcal{S} gives vk and the generated signatures to \mathcal{F} . Because vk and the signatures are distributed within $\text{negl}(n)$ statistical distance of those in the real attack (for any choice of the prefix p), with probability at least $\delta/2 - \text{negl}(n)$, \mathcal{F} outputs a forgery (μ^*, \mathbf{v}^*) where μ^* is different from all the queried messages, $\mathbf{A}_{\mu^*} \mathbf{v}^* = \mathbf{u}$, and $\|\mathbf{v}^*\| \leq s \cdot \sqrt{m}$. Furthermore, conditioned on this event, μ^* has p as a prefix with probability at least $1/((\ell-1)Q+1) - \text{negl}(n)$, because p is still essentially uniform in P conditioned on the view of \mathcal{F} . Therefore, all of these events occur with probability at least $\delta/(2(\ell-1)Q+2) - \text{negl}(n)$.

In such a case, \mathcal{S} extracts a solution to its SIS challenge instance from the forgery (μ^*, \mathbf{v}^*) as follows. Because μ^* starts with p , we have $\mathbf{A}_{\mu^*} = [\bar{\mathbf{A}} \mid \mathbf{B} \mid -\bar{\mathbf{A}}\mathbf{R}^*]$ for $\mathbf{R}^* = \mathbf{R}_0 + \sum_{i \in [\ell]} \mu_i^* \mathbf{R}_i$, and so

$$\underbrace{[\bar{\mathbf{A}} \mid \mathbf{B}]}_{\mathbf{A}} \underbrace{\begin{bmatrix} \mathbf{I}_{\bar{m}} & -\mathbf{R}^* \\ & \mathbf{I}_{nk} \end{bmatrix}}_{\mathbf{z}} \mathbf{v}^* = \mathbf{u} \bmod q,$$

as desired. Because $\|\mathbf{v}^*\| \leq s \cdot \sqrt{m} = O(\sqrt{\ell nk}) \cdot \omega(\sqrt{\log n})^2$ and $s_1(\mathbf{R}^*) = \sqrt{\ell+1} \cdot O(\sqrt{m} + \sqrt{nk}) \cdot \omega(\sqrt{\log n})$ with overwhelming probability (conditioned on the view of \mathcal{F} and any fixed \mathbf{H}_i), we have $\|\mathbf{z}\| = O(\ell(nk)^{3/2}) \cdot \omega(\sqrt{\log n})^3$, which is at most $\beta - 1$, as desired.

Now we consider an \mathcal{F} that forges on one of its queried messages (with probability at least $\delta/2$). Our reduction \mathcal{S} simulates the attack to \mathcal{F} as follows:

- Invoke \mathcal{F} to receive up to Q distinct messages $\mu^{(1)}, \mu^{(2)}, \dots \in \{0, 1\}^\ell$. Choose one of these messages $\mu = \mu^{(i)}$ uniformly at random, “guessing” that the eventual forgery will be on μ .
- Construct a verification key $vk = (\mathbf{A}, \mathbf{A}_0, \dots, \mathbf{A}_\ell, \mathbf{u})$: generate \mathbf{A}_i exactly as above, using $p = \mu$. Then choose $\mathbf{v} \leftarrow D_{\mathbb{Z}^m, s}$ and let $\mathbf{u} = \mathbf{A}_\mu \mathbf{v}$, where \mathbf{A}_μ is defined in the usual way.
- Generate signatures for the queried messages: for all the queries except μ , proceed exactly as above (which is possible because all the queries are distinct and hence do not have $p = \mu$ as a prefix). For μ , use \mathbf{v} as the signature, which has the required distribution $D_{\Lambda_{\mathbf{u}}^\perp(\mathbf{A}_\mu), s}$ by construction.

When \mathcal{S} gives vk and the signatures to \mathcal{F} , with probability at least $\delta/2 - \text{negl}(n)$ the forger must output a forgery (μ^*, \mathbf{v}^*) where μ^* is one of its queries, \mathbf{v}^* is different from the corresponding signature it received, $\mathbf{A}_{\mu^*} \mathbf{v}^* = \mathbf{u}$, and $\|\mathbf{v}^*\| \leq s \cdot \sqrt{m}$. Because vk and the signatures are appropriately distributed for any

choice μ that \mathcal{S} made, conditioned on the above event the probability that $\mu^* = \mu$ is at least $1/Q - \text{negl}(n)$. Therefore, all of these events occur with probability at least $\delta/(2Q) - \text{negl}(n)$.

In such a case, \mathcal{S} extracts a solution to its SIS challenge from the forgery as follows. Because $\mu^* = \mu$, we have $\mathbf{A}_{\mu^*} = [\bar{\mathbf{A}} \mid \mathbf{B} \mid -\bar{\mathbf{A}}\mathbf{R}^*]$ for $\mathbf{R}^* = \mathbf{R}_0 + \sum_{i \in [\ell]} \mu_i^* \mathbf{R}_i$, and so

$$\underbrace{[\bar{\mathbf{A}} \mid \mathbf{B}]}_{\mathbf{A}} \underbrace{\begin{bmatrix} \mathbf{I}_{\bar{m}} & -\mathbf{R}^* \\ & \mathbf{I}_{nk} \end{bmatrix}}_{\mathbf{z}} (\mathbf{v}^* - \mathbf{v}) = \mathbf{0} \bmod q.$$

Because both $\|\mathbf{v}^*\|, \|\mathbf{v}\| \leq s \cdot \sqrt{m} = O(\sqrt{\ell nk}) \cdot \omega(\sqrt{\log n})^2$ and $s_1(\mathbf{R}^*) = O(\sqrt{\ell nk}) \cdot \omega(\sqrt{\log n})$ with overwhelming probability (conditioned on the view of \mathcal{F} and any fixed \mathbf{H}_i), we have $\|\mathbf{z}\| = O(\ell(nk)^{3/2}) \cdot \omega(\sqrt{\log n})^3$ with overwhelming probability, as needed. It just remains to show that $\mathbf{z} \neq \mathbf{0}$ with overwhelming probability. To see this, write $\mathbf{w} = \mathbf{v}^* - \mathbf{v} = (\mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3) \in \mathbb{Z}^{\bar{m}} \times \mathbb{Z}^{nk} \times \mathbb{Z}^{nk}$, with $\mathbf{w} \neq \mathbf{0}$. If $\mathbf{w}_2 \neq \mathbf{0}$ or $\mathbf{w}_3 = \mathbf{0}$, then $\mathbf{z} \neq \mathbf{0}$ and we are done. Otherwise, choose some entry of \mathbf{w}_3 that is nonzero; without loss of generality say it is w_m . Let $\mathbf{r} = (\mathbf{R}_0)_{nk}$. Now for any fixed values of \mathbf{R}_i for $i \in [\ell]$ and fixed first $nk - 1$ columns of \mathbf{R}_0 , we have $\mathbf{z} = \mathbf{0}$ only if $\mathbf{r} \cdot \mathbf{w}_m = \mathbf{y} \in \mathbb{R}^{\bar{m}}$ for some fixed \mathbf{y} . Conditioned on the adversary's view (specifically, $(\mathbf{A}_0)_{nk} = \bar{\mathbf{A}}\mathbf{r}$), \mathbf{r} is distributed as a discrete Gaussian of parameter $\geq 2\eta_\epsilon(\Lambda^\perp(\bar{\mathbf{A}}))$ for some $\epsilon = \text{negl}(n)$ over a coset of $\Lambda^\perp(\bar{\mathbf{A}})$. Then by Lemma 2.7, we have $\mathbf{r} = \mathbf{y}/w_m$ with only $2^{-\Omega(n)}$ probability, and we are done. \square

6.3 Chosen Ciphertext-Secure Encryption

Definitions. A public-key cryptosystem for a message space \mathcal{M} (which may depend on the security parameter) is a tuple of algorithms as follows:

- $\text{Gen}(1^n)$ outputs a public encryption key pk and a secret decryption key sk .
- $\text{Enc}(pk, m)$, given a public key pk and a message $m \in \mathcal{M}$, outputs a ciphertext $c \in \{0, 1\}^*$.
- $\text{Dec}(sk, c)$, given a decryption key sk and a ciphertext c , outputs some $m \in \mathcal{M} \cup \{\perp\}$.

The correctness requirement is: for any $m \in \mathcal{M}$, generate $(pk, sk) \leftarrow \text{Gen}(1^n)$ and $c \leftarrow \text{Enc}(pk, m)$. Then $\text{Dec}(sk, c)$ should output m with overwhelming probability (over all the randomness in the experiment).

We recall the two notions of security under chosen-ciphertext attacks. We start with the weaker notion of CCA1 (or “lunchtime”) security. Let \mathcal{A} be any nonuniform probabilistic polynomial-time algorithm. First, we generate $(pk, sk) \leftarrow \text{Gen}(1^n)$ and give pk to \mathcal{A} . Next, we give \mathcal{A} oracle access to the decryption procedure $\text{Dec}(sk, \cdot)$. Next, \mathcal{A} outputs two messages $m_0, m_1 \in \mathcal{M}$ and is given a challenge ciphertext $c \leftarrow \text{Enc}(pk, m_b)$ for either $b = 0$ or $b = 1$. The scheme is CCA1-secure if the views of \mathcal{A} (i.e., the public key pk , the answers to its oracle queries, and the ciphertext c) for $b = 0$ versus $b = 1$ are computationally indistinguishable (i.e., \mathcal{A} 's acceptance probabilities for $b = 0$ versus $b = 1$ differ by only $\text{negl}(n)$). In the stronger CCA2 notion, after receiving the challenge ciphertext, \mathcal{A} continues to have access to the decryption oracle $\text{Dec}(sk, \cdot)$ for any query not equal to the challenge ciphertext c ; security is defined similarly.

Construction. To highlight the main new ideas, here we present a public-key encryption scheme that is CCA1-secure. Full CCA2 security can be obtained via relatively generic transformations using either strongly unforgeable one-time signatures [DDN00], or a message authentication code and weak form of commitment [BCHK07]; we omit these details.

Our scheme involves a number of parameters, for which we give some exemplary asymptotic bounds. In what follows, $\omega(\sqrt{\log n})$ represents a fixed function that asymptotically grows faster than $\sqrt{\log n}$.

- $\mathbf{G} \in \mathbb{Z}_q^{n \times nk}$ is a gadget matrix for large enough prime power $q = p^e = \text{poly}(n)$ and $k = O(\log q) = O(\log n)$. We require an oracle \mathcal{O} that solves LWE with respect to $\Lambda(\mathbf{G}^t)$ for any error vector in some $\mathcal{P}_{1/2}(q \cdot \mathbf{B}^{-t})$ where $\|\mathbf{B}\| = O(1)$. (See for example the constructions from Section 4.)
- $\bar{m} = O(nk)$ and $\mathcal{D} = D_{\mathbb{Z}, \omega(\sqrt{\log n})}^{\bar{m} \times nk}$ so that $(\bar{\mathbf{A}}, \bar{\mathbf{A}}\mathbf{R})$ is $\text{negl}(n)$ -far from uniform for $\bar{\mathbf{A}} \leftarrow \mathbb{Z}_q^{n \times \bar{m}}$ and $\mathbf{R} \leftarrow \mathcal{D}$, and $m = \bar{m} + nk$ is the total dimension of the public key and ciphertext.
- α is an error rate for LWE, for sufficiently large $1/\alpha = O(nk) \cdot \omega(\sqrt{\log n})$.

Our scheme requires a special collection of elements in the ring $\mathcal{R} = \mathbb{Z}_q[x]/(f(x))$ as constructed in Section 6.1 (recall that here $q = p^e$). We need a very large set $\mathcal{U} = \{u_1, \dots, u_\ell\} \subset \mathcal{R}$ with the “unit differences” property: for any $i \neq j$, the difference $u_i - u_j \in \mathcal{R}^*$, and hence $h(u_i - u_j) = h(u_i) - h(u_j) \in \mathbb{Z}_q^{n \times n}$ is invertible. (Note that the u_i s need not all be units themselves.) Concretely, by the characterization of units in \mathcal{R} given above, we take \mathcal{U} to be all linear combinations of the monomials $1, x, \dots, x^{n-1}$ with coefficients in $\{0, \dots, p-1\}$, of which there are exactly p^n . Since the difference between any two such distinct elements is nonzero modulo p , it is a unit.

The system has message space $\{0, 1\}^{nk}$, which we map bijectively to the cosets of $\Lambda/2\Lambda$ for $\Lambda = \Lambda(\mathbf{G}^t)$ via some function encode that is efficient to evaluate and invert. Concretely, letting $\mathbf{S} \in \mathbb{Z}^{nk \times nk}$ be any basis of Λ , we can map $\mathbf{m} \in \{0, 1\}^{nk}$ to $\text{encode}(\mathbf{m}) = \mathbf{S}\mathbf{m} \in \mathbb{Z}^{nk}$.

- $\text{Gen}(1^n)$: choose $\bar{\mathbf{A}} \leftarrow \mathbb{Z}_q^{n \times \bar{m}}$ and $\mathbf{R} \leftarrow \mathcal{D}$, letting $\mathbf{A}_1 = -\bar{\mathbf{A}}\mathbf{R} \bmod q$. The public key is $pk = \mathbf{A} = [\bar{\mathbf{A}} \mid \mathbf{A}_1] \in \mathbb{Z}_q^{n \times m}$ and the secret key is $sk = \mathbf{R}$.
- $\text{Enc}(pk = [\bar{\mathbf{A}} \mid \mathbf{A}_1], \mathbf{m} \in \{0, 1\}^{nk})$: choose nonzero $u \leftarrow \mathcal{U}$ and let $\mathbf{A}_u = [\bar{\mathbf{A}} \mid \mathbf{A}_1 + h(u)\mathbf{G}]$. Choose $\mathbf{s} \leftarrow \mathbb{Z}_q^n$, $\bar{\mathbf{e}} \leftarrow D_{\mathbb{Z}, \alpha q}^{\bar{m}}$, and $\mathbf{e}_1 \leftarrow D_{\mathbb{Z}, s}^{nk}$ where $s^2 = (\|\bar{\mathbf{e}}\|^2 + \bar{m}(\alpha q)^2) \cdot \omega(\sqrt{\log n})^2$.

Let

$$\mathbf{b}^t = 2(\mathbf{s}^t \mathbf{A}_u \bmod q) + \mathbf{e}^t + (\mathbf{0}, \text{encode}(\mathbf{m}))^t \bmod 2q,$$

where $\mathbf{e} = (\bar{\mathbf{e}}, \mathbf{e}_1) \in \mathbb{Z}^m$ and $\mathbf{0}$ has dimension \bar{m} . (Note the use of mod- $2q$ arithmetic: $2(\mathbf{s}^t \mathbf{A}_u \bmod q)$ is an element of the lattice $2\Lambda(\mathbf{A}_u^t) \supseteq 2q\mathbb{Z}^m$.) Output the ciphertext $c = (u, \mathbf{b}) \in \mathcal{U} \times \mathbb{Z}_{2q}^m$.

- $\text{Dec}(sk = \mathbf{R}, c = (u, \mathbf{b}) \in \mathcal{U} \times \mathbb{Z}_{2q}^m)$: Let $\mathbf{A}_u = [\bar{\mathbf{A}} \mid \mathbf{A}_1 + h(u)\mathbf{G}] = [\bar{\mathbf{A}} \mid h(u)\mathbf{G} - \bar{\mathbf{A}}\mathbf{R}]$.
 1. If c does not parse or $u = 0$, output \perp . Otherwise, call $\text{Invert}^{\mathcal{O}}(\mathbf{R}, \mathbf{A}_u, \mathbf{b} \bmod q)$ to get values $\mathbf{z} \in \mathbb{Z}_q^n$ and $\mathbf{e} = (\bar{\mathbf{e}}, \mathbf{e}_1) \in \mathbb{Z}^{\bar{m}} \times \mathbb{Z}^{nk}$ for which $\mathbf{b}^t = \mathbf{z}^t \mathbf{A}_u + \mathbf{e}^t \bmod q$. (Note that $h(u) \in \mathbb{Z}_q^{n \times n}$ is invertible, as required by Invert .) If the call to Invert fails for any reason, output \perp .
 2. If $\|\bar{\mathbf{e}}\| \geq \alpha q \sqrt{\bar{m}}$ or $\|\mathbf{e}_1\| \geq \alpha q \sqrt{2\bar{m}nk} \cdot \omega(\sqrt{\log n})$, output \perp .
 3. Let $\mathbf{v} = \mathbf{b} - \mathbf{e} \bmod 2q$, parsed as $\mathbf{v} = (\bar{\mathbf{v}}, \mathbf{v}_1) \in \mathbb{Z}_{2q}^{\bar{m}} \times \mathbb{Z}_{2q}^{nk}$. If $\bar{\mathbf{v}} \notin 2\Lambda(\bar{\mathbf{A}}^t)$, output \perp . Finally, output $\text{encode}^{-1}(\mathbf{v}^t \begin{bmatrix} \mathbf{R} \\ \mathbf{I} \end{bmatrix} \bmod 2q) \in \{0, 1\}^{nk}$ if it exists, otherwise output \perp .

(In practice, to avoid timing attacks one would perform all of the Dec operations first, and only then finally output \perp if any of the validity tests failed.)

Lemma 6.2. *The above scheme has only $2^{-\Omega(n)}$ probability of decryption error.*

The error probability can be made zero by changing Gen and Enc so that they resample \mathbf{R} , $\bar{\mathbf{e}}$, and/or \mathbf{e}_1 in the rare event that they violate the corresponding bounds given in the proof below.

Proof. Let $(\mathbf{A}, \mathbf{R}) \leftarrow \text{Gen}(1^n)$. By Lemma 2.9, we have $s_1(\mathbf{R}) \leq O(\sqrt{nk}) \cdot \omega(\sqrt{\log n})$ except with probability $2^{-\Omega(n)}$. Now consider the random choices made by $\text{Enc}(\mathbf{A}, \mathbf{m})$ for arbitrary $\mathbf{m} \in \{0, 1\}^{nk}$. By Lemma 2.6, we have both $\|\bar{\mathbf{e}}\| < \alpha q \sqrt{m}$ and $\|\mathbf{e}_1\| < \alpha q \sqrt{2mnk} \cdot \omega(\sqrt{\log n})$, except with probability $2^{-\Omega(n)}$. Letting $\mathbf{e} = (\bar{\mathbf{e}}, \mathbf{e}_1)$, we have

$$\|\mathbf{e}^t \begin{bmatrix} \mathbf{R} \\ \mathbf{I} \end{bmatrix}\| \leq \|\bar{\mathbf{e}}^t \mathbf{R}\| + \|\mathbf{e}_1\| < \alpha q \cdot O(nk) \cdot \omega(\sqrt{\log n}).$$

In particular, for large enough $1/\alpha = O(nk) \cdot \omega(\sqrt{\log n})$ we have $\mathbf{e}^t \begin{bmatrix} \mathbf{R} \\ \mathbf{I} \end{bmatrix} \in \mathcal{P}_{1/2}(q \cdot \mathbf{B}^{-t})$. Therefore, the call to Invert made by $\text{Dec}(\mathbf{R}, (u, \mathbf{b}))$ returns \mathbf{e} . It follows that for $\mathbf{v} = (\bar{\mathbf{v}}, \mathbf{v}_1) = \mathbf{b} - \mathbf{e} \bmod 2q$, we have $\bar{\mathbf{v}} \in 2\Lambda(\bar{\mathbf{A}}^t)$ as needed. Finally,

$$\mathbf{v}^t \begin{bmatrix} \mathbf{R} \\ \mathbf{I} \end{bmatrix} = 2(\mathbf{s}^t h(u) \mathbf{G} \bmod q) + \text{encode}(\mathbf{m}) \bmod 2q,$$

which is in the coset $\text{encode}(\mathbf{m}) \in \Lambda(\mathbf{G}^t)/2\Lambda(\mathbf{G}^t)$, and so Dec outputs \mathbf{m} as desired. \square

Theorem 6.3. *The above scheme is CCA1-secure assuming the hardness of decision-LWE $_{q, \alpha'}$ for $\alpha' = \alpha/3 \geq 2\sqrt{n}/q$.*

Proof. We start by giving a particular form of discretized LWE that we will need below. Given access to an LWE distribution $A_{\mathbf{s}, \alpha'}$ over $\mathbb{Z}_q^n \times \mathbb{T}$ for any $\mathbf{s} \in \mathbb{Z}_q^n$ (where recall that $\mathbb{T} = \mathbb{R}/\mathbb{Z}$), by [Pei10, Theorem 3.1] we can transform its samples $(\mathbf{a}, b = \langle \mathbf{s}, \mathbf{a} \rangle / q + e \bmod 1)$ to have the form $(\mathbf{a}, 2(\langle \mathbf{s}, \mathbf{a} \rangle \bmod q) + \mathbf{e}' \bmod 2q)$ for $\mathbf{e}' \leftarrow D_{\mathbb{Z}, \alpha q}$, by mapping $b \mapsto 2qb + D_{\mathbb{Z}-2qb, s} \bmod 2q$ where $s^2 = (\alpha q)^2 - (2\alpha' q)^2 \geq 4n \geq \eta_\epsilon(\mathbb{Z})^2$. This transformation maps the uniform distribution over $\mathbb{Z}_q^n \times \mathbb{T}$ to the uniform distribution over $\mathbb{Z}_q^n \times \mathbb{Z}_{2q}$, so the discretized distribution is pseudorandom under the hypothesis of the theorem.

We proceed via a sequence of hybrid games. The game H_0 is exactly the CCA1 attack with the system described above.

In game H_1 , we change how the public key \mathbf{A} and challenge ciphertext $c^* = (u^*, \mathbf{b}^*)$ are constructed, and the way that decryption queries are answered (slightly), but in a way that introduces only $\text{negl}(n)$ statistical difference with H_0 . At the start of the experiment we choose nonzero $u^* \leftarrow \mathcal{U}$ and let the public key be $\mathbf{A} = [\bar{\mathbf{A}} \mid \mathbf{A}_1] = [\bar{\mathbf{A}} \mid -h(u^*)\mathbf{G} - \bar{\mathbf{A}}\mathbf{R}]$, where $\bar{\mathbf{A}}$ and \mathbf{R} are chosen in the same way as in H_0 . (In particular, we still have $s_1(\mathbf{R}) \leq O(\sqrt{nk}) \cdot \omega(\sqrt{\log n})$ with overwhelming probability.) Note that \mathbf{A} is still $\text{negl}(n)$ -uniform for any choice of u^* , so conditioned on any fixed choice of \mathbf{A} , the value of u^* is statistically hidden from the attacker. To aid with decryption queries, we also choose an arbitrary (not necessarily short) $\hat{\mathbf{R}} \in \mathbb{Z}^{\bar{m} \times nk}$ such that $\mathbf{A}_1 = -\bar{\mathbf{A}}\hat{\mathbf{R}} \bmod q$.

To answer a decryption query on a ciphertext (u, \mathbf{b}) , we use an algorithm very similar to Dec with trapdoor \mathbf{R} . After testing whether $u = 0$ (and outputting \perp if so), we call $\text{Invert}^O(\mathbf{R}, \mathbf{A}_u, \mathbf{b} \bmod q)$ to get some $\mathbf{z} \in \mathbb{Z}_q^n$ and $\mathbf{e} \in \mathbb{Z}^m$, where

$$\mathbf{A}_u = [\bar{\mathbf{A}} \mid \mathbf{A}_1 + h(u)\mathbf{G}] = [\bar{\mathbf{A}} \mid h(u - u^*)\mathbf{G} - \bar{\mathbf{A}}\mathbf{R}].$$

(If Invert fails, we output \perp .) We then perform steps 2 and 3 on $\mathbf{e} \in \mathbb{Z}^m$ and $\mathbf{v} = \mathbf{b} - \mathbf{e} \bmod 2q$ exactly as in Dec , except that we use $\hat{\mathbf{R}}$ in place of \mathbf{R} when decoding the message in step 3.

We now analyze the behavior of this decryption routine. Whenever $u \neq u^*$, which is the case with overwhelming probability because u^* is statistically hidden, by the “unit differences” property on \mathcal{U} we have that $h(u - u^*) \in \mathbb{Z}_q^{n \times n}$ is invertible, as required by the call to Invert . Now, either there exists an \mathbf{e} that satisfies the validity tests in step 2 and such that $\mathbf{b}^t = \mathbf{z}^t \mathbf{A}_u + \mathbf{e}^t \bmod q$ for some $\mathbf{z} \in \mathbb{Z}_q^n$, or there does not. In the latter case, no matter what Invert does in H_0 and H_1 , step 2 will return \perp in both games. Now consider the former case: by the constraints on \mathbf{e} , we have $\mathbf{e}^t \begin{bmatrix} \mathbf{R} \\ \mathbf{I} \end{bmatrix} \in \mathcal{P}_{1/2}(q \cdot \mathbf{B}^{-t})$ in both games, so the call to Invert

must return this \mathbf{e} (but possibly different \mathbf{z}) in both games. Finally, the result of decryption is the same in both games: if $\bar{\mathbf{v}} \in 2\Lambda(\bar{\mathbf{A}}^t)$ (otherwise, both games return \perp), then we can express \mathbf{v} as

$$\mathbf{v}^t = 2(\mathbf{s}^t \mathbf{A}_u \bmod q) + (\mathbf{0}, \mathbf{v}')^t \bmod 2q$$

for some $\mathbf{s} \in \mathbb{Z}_q^n$ and $\mathbf{v}' \in \mathbb{Z}_{2q}^{nk}$. Then for *any* solution $\mathbf{R} \in \mathbb{Z}^{\bar{m} \times nk}$ to $\mathbf{A}_1 = -\bar{\mathbf{A}}\mathbf{R} \bmod q$, we have

$$\mathbf{v}^t \begin{bmatrix} \mathbf{R} \\ \mathbf{I} \end{bmatrix} = 2(\mathbf{s}^t h(u) \mathbf{G} \bmod q) + (\mathbf{v}')^t \bmod 2q.$$

In particular, this holds for the \mathbf{R} in H_0 and the $\hat{\mathbf{R}}$ in H_1 that are used for decryption. It follows that both games output $\text{encode}^{-1}(\mathbf{v}')$, if it exists (and \perp otherwise).

Finally, in H_1 we produce the challenge ciphertext (u, \mathbf{b}) on a message $\mathbf{m} \in \{0, 1\}^{nk}$ as follows. Let $u = u^*$, and choose $\mathbf{s} \leftarrow \mathbb{Z}_q^n$ and $\bar{\mathbf{e}} \leftarrow D_{\mathbb{Z}, \alpha q}^{\bar{m}}$ as usual, but do not choose \mathbf{e}_1 . Note that $\mathbf{A}_u = [\bar{\mathbf{A}} \mid -\bar{\mathbf{A}}\mathbf{R}]$. Let $\bar{\mathbf{b}}^t = 2(\mathbf{s}^t \bar{\mathbf{A}} \bmod q) + \bar{\mathbf{e}}^t \bmod 2q$. Let

$$\mathbf{b}_1^t = -\bar{\mathbf{b}}^t \mathbf{R} + \hat{\mathbf{e}}^t + \text{encode}(\mathbf{m}) \bmod 2q,$$

where $\hat{\mathbf{e}} \leftarrow D_{\mathbb{Z}, \alpha q \sqrt{m} \cdot \omega(\sqrt{\log n})}^{nk}$, and output $(u, \mathbf{b} = (\bar{\mathbf{b}}, \mathbf{b}_1))$. We now show that the distribution of (u, \mathbf{b}) is within $\text{negl}(n)$ statistical distance of that in H_0 , given the attacker's view (i.e., pk and the results of the decryption queries). Clearly, u and $\bar{\mathbf{b}}$ have essentially the same distribution as in H_0 , because u is $\text{negl}(n)$ -uniform given pk , and by construction of $\bar{\mathbf{b}}$. By substitution, we have

$$\mathbf{b}_1^t = 2(\mathbf{s}^t (-\bar{\mathbf{A}}\mathbf{R}) \bmod q) + (\bar{\mathbf{e}}^t \mathbf{R} + \hat{\mathbf{e}}^t) + \text{encode}(\mathbf{m}).$$

Therefore, it suffices to show that for fixed $\bar{\mathbf{e}}$, each $\langle \bar{\mathbf{e}}, \mathbf{r}_i \rangle + \hat{e}_i$ has distribution $\text{negl}(n)$ -far from $D_{\mathbb{Z}, s}$, where $s^2 = (\|\bar{\mathbf{e}}\|^2 + m(\alpha q)^2) \cdot \omega(\sqrt{\log n})^2$, over the random choice of \mathbf{r}_i (conditioned on the value of $\bar{\mathbf{A}}\mathbf{r}_i$ from the public key) and of \hat{e}_i . Because each \mathbf{r}_i is an independent discrete Gaussian over a coset of $\Lambda^\perp(\bar{\mathbf{A}})$, the claim follows essentially by [Reg05, Corollary 3.10], but adapted to discrete random variables using [Pei10, Theorem 3.1] in place of [Reg05, Claim 3.9].

In game H_2 , we only change how the $\bar{\mathbf{b}}$ component of the challenge ciphertext is created, letting it be uniformly random in $\mathbb{Z}_{2q}^{\bar{m}}$. We construct pk , answer decryption queries, and construct \mathbf{b}_1 in exactly the same way as in H_1 . First observe that under our (discretized) LWE hardness assumption, games H_1 and H_2 are computationally indistinguishable by an elementary reduction: given $(\bar{\mathbf{A}}, \bar{\mathbf{b}}) \in \mathbb{Z}_q^{n \times \bar{m}} \times \mathbb{Z}_{2q}^{\bar{m}}$ where $\bar{\mathbf{A}}$ is uniformly random and either $\bar{\mathbf{b}}^t = 2(\mathbf{s}^t \bar{\mathbf{A}} \bmod q) + \bar{\mathbf{e}}^t \bmod 2q$ (for $\mathbf{s} \leftarrow \mathbb{Z}_q^n$ and $\bar{\mathbf{e}} \leftarrow D_{\mathbb{Z}, \alpha q}^{\bar{m}}$) or $\bar{\mathbf{b}}$ is uniformly random, we can efficiently emulate either game H_1 or H_2 (respectively) by doing everything exactly as in the two games, except using the given $\bar{\mathbf{A}}$ and $\bar{\mathbf{b}}$ when constructing the public key and challenge ciphertext.

Now by the leftover hash lemma, $(\bar{\mathbf{A}}, \bar{\mathbf{b}}^t, \bar{\mathbf{A}}\mathbf{R}, -\bar{\mathbf{b}}^t \mathbf{R})$ is $\text{negl}(n)$ -uniform when \mathbf{R} is chosen as in H_2 . Therefore, the challenge ciphertext has the same distribution (up to $\text{negl}(n)$ statistical distance) for any encrypted message, and so the adversary's advantage is negligible. This completes the proof. \square

References

- [ABB10a] S. Agrawal, D. Boneh, and X. Boyen. Efficient lattice (H)IBE in the standard model. In *EUROCRYPT*, pages 553–572. 2010.
- [ABB10b] S. Agrawal, D. Boneh, and X. Boyen. Lattice basis delegation in fixed dimension and shorter-ciphertext hierarchical IBE. In *CRYPTO*, pages 98–115. 2010.

- [ACPS09] B. Applebaum, D. Cash, C. Peikert, and A. Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In *CRYPTO*, pages 595–618. 2009.
- [Ajt96] M. Ajtai. Generating hard instances of lattice problems. *Quaderni di Matematica*, 13:1–32, 2004. Preliminary version in STOC 1996.
- [Ajt99] M. Ajtai. Generating hard instances of the short basis problem. In *ICALP*, pages 1–9. 1999.
- [AP09] J. Alwen and C. Peikert. Generating shorter bases for hard random lattices. *Theory of Computing Systems*, 48(3):535–553, April 2011. Preliminary version in STACS 2009.
- [Bab85] L. Babai. On Lovász’ lattice reduction and the nearest lattice point problem. *Combinatorica*, 6(1):1–13, 1986. Preliminary version in STACS 1985.
- [Ban93] W. Banaszczyk. New bounds in some transference theorems in the geometry of numbers. *Mathematische Annalen*, 296(4):625–635, 1993.
- [BCHK07] D. Boneh, R. Canetti, S. Halevi, and J. Katz. Chosen-ciphertext security from identity-based encryption. *SIAM J. Comput.*, 36(5):1301–1328, 2007.
- [BFKL93] A. Blum, M. L. Furst, M. J. Kearns, and R. J. Lipton. Cryptographic primitives based on hard learning problems. In *CRYPTO*, pages 278–291. 1993.
- [BGV11] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. Fully homomorphic encryption without bootstrapping. Cryptology ePrint Archive, Report 2011/277, 2011. <http://eprint.iacr.org/>.
- [Boy10] X. Boyen. Lattice mixing and vanishing trapdoors: A framework for fully secure short signatures and more. In *Public Key Cryptography*, pages 499–517. 2010.
- [BV11a] Z. Brakerski and V. Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In *FOCS*. 2011. To appear.
- [BV11b] Z. Brakerski and V. Vaikuntanathan. Fully homomorphic encryption from ring-LWE and security for key dependent messages. In *CRYPTO*, pages 505–524. 2011.
- [CHKP10] D. Cash, D. Hofheinz, E. Kiltz, and C. Peikert. Bonsai trees, or how to delegate a lattice basis. In *EUROCRYPT*, pages 523–552. 2010.
- [CN11] Y. Chen and P. Q. Nguyen. BKZ 2.0: Simulation and better lattice security estimates. In *ASIACRYPT*. 2011. To appear.
- [DDN00] D. Dolev, C. Dwork, and M. Naor. Nonmalleable cryptography. *SIAM J. Comput.*, 30(2):391–437, 2000.
- [DF94] Y. Desmedt and Y. Frankel. Perfect homomorphic zero-knowledge threshold schemes over any finite abelian group. *SIAM J. Discrete Math.*, 7(4):667–679, 1994.
- [Feh98] S. Fehr. *Span Programs over Rings and How to Share a Secret from a Module*. Master’s thesis, ETH Zurich, Institute for Theoretical Computer Science, 1998.
- [Gen09a] C. Gentry. *A fully homomorphic encryption scheme*. Ph.D. thesis, Stanford University, 2009. crypto.stanford.edu/craig.

- [Gen09b] C. Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178. 2009.
- [GGH97] O. Goldreich, S. Goldwasser, and S. Halevi. Public-key cryptosystems from lattice reduction problems. In *CRYPTO*, pages 112–131. 1997.
- [GH11] C. Gentry and S. Halevi. Fully homomorphic encryption without squashing using depth-3 arithmetic circuits. In *FOCS*. 2011. To appear.
- [GHV10] C. Gentry, S. Halevi, and V. Vaikuntanathan. A simple BGN-type cryptosystem from LWE. In *EUROCRYPT*, pages 506–522. 2010.
- [GKV10] S. D. Gordon, J. Katz, and V. Vaikuntanathan. A group signature scheme from lattice assumptions. In *ASIACRYPT*, pages 395–412. 2010.
- [GN08] N. Gama and P. Q. Nguyen. Predicting lattice reduction. In *EUROCRYPT*, pages 31–51. 2008.
- [GPV08] C. Gentry, C. Peikert, and V. Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC*, pages 197–206. 2008.
- [HW09] S. Hohenberger and B. Waters. Short and stateless signatures from the RSA assumption. In *CRYPTO*, pages 654–670. 2009.
- [Kle00] P. N. Klein. Finding the closest lattice vector when it’s unusually close. In *SODA*, pages 937–941. 2000.
- [KMO10] E. Kiltz, P. Mohassel, and A. O’Neill. Adaptive trapdoor functions and chosen-ciphertext security. In *EUROCRYPT*, pages 673–692. 2010.
- [KR00] H. Krawczyk and T. Rabin. Chameleon signatures. In *NDSS*. 2000.
- [LM06] V. Lyubashevsky and D. Micciancio. Generalized compact knapsacks are collision resistant. In *ICALP (2)*, pages 144–155. 2006.
- [LM08] V. Lyubashevsky and D. Micciancio. Asymptotically efficient lattice-based digital signatures. In *TCC*, pages 37–54. 2008.
- [LM09] V. Lyubashevsky and D. Micciancio. On bounded distance decoding, unique shortest vectors, and the minimum distance problem. In *CRYPTO*, pages 577–594. 2009.
- [LMPR08] V. Lyubashevsky, D. Micciancio, C. Peikert, and A. Rosen. SWIFFT: A modest proposal for FFT hashing. In *FSE*, pages 54–72. 2008.
- [LP11] R. Lindner and C. Peikert. Better key sizes (and attacks) for LWE-based encryption. In *CT-RSA*, pages 319–339. 2011.
- [LPR10] V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. In *EUROCRYPT*, pages 1–23. 2010.
- [Lyu08] V. Lyubashevsky. Lattice-based identification schemes secure under active attacks. In *Public Key Cryptography*, pages 162–179. 2008.

- [MG02] D. Micciancio and S. Goldwasser. *Complexity of Lattice Problems: a cryptographic perspective*, volume 671 of *The Kluwer International Series in Engineering and Computer Science*. Kluwer Academic Publishers, Boston, Massachusetts, 2002.
- [Mic02] D. Micciancio. Generalized compact knapsacks, cyclic lattices, and efficient one-way functions. *Computational Complexity*, 16(4):365–411, 2007. Preliminary version in FOCS 2002.
- [MM11] D. Micciancio and P. Mol. Pseudorandom knapsacks and the sample complexity of LWE search-to-decision reductions. In *CRYPTO*, pages 465–484. 2011.
- [MR04] D. Micciancio and O. Regev. Worst-case to average-case reductions based on Gaussian measures. *SIAM J. Comput.*, 37(1):267–302, 2007. Preliminary version in FOCS 2004.
- [MR09] D. Micciancio and O. Regev. Lattice-based cryptography. In *Post Quantum Cryptography*, pages 147–191. Springer, February 2009.
- [Pei09a] C. Peikert. Bonsai trees (or, arboriculture in lattice-based cryptography). Cryptology ePrint Archive, Report 2009/359, July 2009. <http://eprint.iacr.org/>.
- [Pei09b] C. Peikert. Public-key cryptosystems from the worst-case shortest vector problem. In *STOC*, pages 333–342. 2009.
- [Pei10] C. Peikert. An efficient and parallel Gaussian sampler for lattices. In *CRYPTO*, pages 80–97. 2010.
- [PR06] C. Peikert and A. Rosen. Efficient collision-resistant hashing from worst-case assumptions on cyclic lattices. In *TCC*, pages 145–166. 2006.
- [PV08] C. Peikert and V. Vaikuntanathan. Noninteractive statistical zero-knowledge proofs for lattice problems. In *CRYPTO*, pages 536–553. 2008.
- [PVW08] C. Peikert, V. Vaikuntanathan, and B. Waters. A framework for efficient and composable oblivious transfer. In *CRYPTO*, pages 554–571. 2008.
- [PW08] C. Peikert and B. Waters. Lossy trapdoor functions and their applications. In *STOC*, pages 187–196. 2008.
- [Reg05] O. Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6):1–40, 2009. Preliminary version in STOC 2005.
- [RS10] M. Rückert and M. Schneider. Selecting secure parameters for lattice-based cryptography. Cryptology ePrint Archive, Report 2010/137, 2010. <http://eprint.iacr.org/>.
- [Rüc10] M. Rückert. Strongly unforgeable signatures and hierarchical identity-based signatures from lattices without random oracles. In *PQCrypto*, pages 182–200. 2010.
- [ST01] A. Shamir and Y. Tauman. Improved online/offline signature schemes. In *CRYPTO*, pages 355–367. 2001.
- [Ver11] R. Vershynin. Introduction to the non-asymptotic analysis of random matrices, January 2011. Available at <http://www-personal.umich.edu/~romanv/papers/non-asymptotic-rmt-plain.pdf>, last accessed 4 Feb 2011.

- [vGHV10] M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan. Fully homomorphic encryption over the integers. In *EUROCRYPT*, pages 24–43. 2010.
- [Wat05] B. Waters. Efficient identity-based encryption without random oracles. In *EUROCRYPT*, pages 114–127. 2005.

Homomorphic Evaluation of the AES Circuit

Craig Gentry
IBM Research

Shai Halevi
IBM Research

Nigel P. Smart
University of Bristol

June 15, 2012

Abstract

We describe a working implementation of leveled homomorphic encryption (without bootstrapping) that can evaluate the AES-128 circuit in three different ways. One variant takes under over 36 hours to evaluate an entire AES encryption operation, using NTL (over GMP) as our underlying software platform, and running on a large-memory machine. Using SIMD techniques, we can process over 54 blocks in each evaluation, yielding an amortized rate of just under 40 minutes per block. Another implementation takes just over two and a half days to evaluate the AES operation, but can process 720 blocks in each evaluation, yielding an amortized rate of just over five minutes per block. We also detail a third implementation, which theoretically could yield even better amortized complexity, but in practice turns out to be less competitive.

For our implementations we develop both AES-specific optimizations as well as several “generic” tools for FHE evaluation. These last tools include (among others) a different variant of the Brakerski-Vaikuntanathan key-switching technique that does not require reducing the norm of the ciphertext vector, and a method of implementing the Brakerski-Gentry-Vaikuntanathan modulus-switching transformation on ciphertexts in CRT representation.

Keywords. AES, Fully Homomorphic Encryption, Implementation

The first and second authors are sponsored by DARPA under agreement number FA8750-11-C-0096. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the U.S. Government. Distribution Statement “A” (Approved for Public Release, Distribution Unlimited).

The third author is sponsored by DARPA and AFRL under agreement number FA8750-11-2-0079. The same disclaimers as above apply. He is also supported by the European Commission through the ICT Programme under Contract ICT-2007-216676 ECRYPT II and via an ERC Advanced Grant ERC-2010-AdG-267188-CRIPTO, by EPSRC via grant COED-EP/I03126X, and by a Royal Society Wolfson Merit Award. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the European Commission or EPSRC.

Contents

1	Introduction	1
2	Background	3
2.1	Notations and Mathematical Background	3
2.2	BGV-type Cryptosystems	3
2.3	Computing on Packed Ciphertexts	6
3	General-Purpose Optimizations	6
3.1	A New Variant of Key Switching	6
3.2	Modulus Switching in Evaluation Representation	8
3.3	Dynamic Noise Management	8
3.4	Randomized Multiplication by Constants	9
4	Homomorphic Evaluation of AES	9
4.1	Homomorphic Evaluation of the Basic Operations	10
4.1.1	AddKey and SubBytes	10
4.1.2	ShiftRows and MixColumns	11
4.1.3	The Cost of One Round Function	12
4.2	Byte- and Bit-Slice Implementations	12
4.3	Performance Details	12
	References	13
A	More Details	15
A.1	Plaintext Slots	15
A.2	Canonical Embedding Norm	15
A.3	Double CRT Representation	16
A.4	Sampling From A_q	17
A.5	Canonical embedding norm of random polynomials	17
B	The Basic Scheme	18
B.1	Our Moduli Chain	18
B.2	Modulus Switching	18
B.3	Key Switching	19
B.4	Key-Generation, Encryption, and Decryption	21
B.5	Homomorphic Operations	22
C	Security Analysis and Parameter Settings	23
C.1	Lower-Bounding the Dimension	23
C.1.1	LWE with Sparse Key	25
C.2	The Modulus Size	25
C.3	Putting It Together	27
D	Scale(c, q_t, q_{t-1}) in dble-CRT Representation	28

1 Introduction

In his breakthrough result [13], Gentry demonstrated that fully-homomorphic encryption was theoretically possible, assuming the hardness of some problems in integer lattices. Since then, many different improvements have been made, for example authors have proposed new variants, improved efficiency, suggested other hardness assumptions, etc. Some of these works were accompanied by implementation [26, 14, 8, 27, 19, 9], but all the implementations so far were either “proofs of concept” that can compute only one basic operation at a time (at great cost), or special-purpose implementations limited to evaluating very simple functions. In this work we report on the first implementation powerful enough to support an “interesting real world circuit”. Specifically, we implemented a variant of the leveled FHE-without-bootstrapping scheme of Brakerski, Gentry, and Vaikuntanathan [5] (BGV), with support for deep enough circuits so that we can evaluate an entire AES-128 encryption operation.

Why AES? We chose to shoot for an evaluation of AES since it seems like a natural benchmark: AES is widely deployed and used extensively in security-aware applications (so it is “practically relevant” to implement it), and the AES circuit is nontrivial on one hand, but on the other hand not astronomical. Moreover the AES circuit has a regular (and quite “algebraic”) structure, which is amenable to parallelism and optimizations. Indeed, for these same reasons AES is often used as a benchmark for implementations of protocols for secure multi-party computation (MPC), for example [24, 10, 17, 18]. Using the same yardstick to measure FHE and MPC protocols is quite natural, since these techniques target similar application domains and in some cases both techniques can be used to solve the same problem.

Beyond being a natural benchmark, homomorphic evaluation of AES decryption also has interesting applications: When data is encrypted under AES and we want to compute on that data, then homomorphic AES decryption would transform this AES-encrypted data into an FHE-encrypted data, and then we could perform whatever computation we wanted. (Such applications were alluded to in [19, 27, 6]).

Why BGV? Our implementation is based on the (ring-LWE-based) BGV cryptosystem [5], which at present is one of three variants that seem the most likely to yield “somewhat practical” homomorphic encryption. The other two are the NTRU-like cryptosystem of López-Alt et al. [21] and the ring-LWE-based fixed-modulus cryptosystem of Brakerski [4]. (These two variants were not yet available when we started our implementation effort.) These three different variants offer somewhat different implementation trade-offs, but they all have similar performance characteristics. At present we do not know which of them will end up being faster in practice, but the differences are unlikely to be very significant. Moreover, we note that most of our optimizations for BGV are useful also for the other two variants.

Our Contributions. Our implementation is based on a variant of the BGV scheme [5, 7, 6] (based on ring-LWE [22]), using the techniques of Smart and Vercauteren (SV) [27] and Gentry, Halevi and Smart (GHS) [15], and we introduce many new optimizations. Some of our optimizations are specific to AES, these are described in Section 4. Most of our optimization, however, are more general-purpose and can be used for homomorphic evaluation of other circuits, these are described in Section 3.

Many of our general-purpose optimizations are aimed at reducing the number of FFTs and CRTs that we need to perform, by reducing the number of times that we need to convert polynomials between coefficient and evaluation representations. Since the cryptosystem is defined over a polynomial ring, many of the operations involve various manipulation of integer polynomials, such as modular multiplications and additions and Frobenius maps. Most of these operations can be performed more efficiently in evaluation

representation, when a polynomial is represented by the vector of values that it assumes in all the roots of the ring polynomial (for example polynomial multiplication is just point-wise multiplication of the evaluation values). On the other hand some operations in BGV-type cryptosystems (such as key switching and modulus switching) seem to require coefficient representation, where a polynomial is represented by listing all its coefficients.¹ Hence a “naive implementation” of FHE would need to convert the polynomials back and forth between the two representations, and these conversions turn out to be the most time-consuming part of the execution. In our implementation we keep ciphertexts in evaluation representation at all times, converting to coefficient representation only when needed for some operation, and then converting back.

We describe variants of key switching and modulus switching that can be implemented while keeping almost all the polynomials in evaluation representation. Our key-switching variant has another advantage, in that it significantly reduces the size of the key-switching matrices in the public key. This is particularly important since the main limiting factor for evaluating deep circuits turns out to be the ability to keep the key-switching matrices in memory. Other optimizations that we present are meant to reduce the number of modulus switching and key switching operations that we need to do. This is done by tweaking some operations (such as multiplication by constant) to get a slower noise increase, by “batching” some operations before applying key switching, and by attaching to each ciphertext an estimate of the “noisiness” of this ciphertext, in order to support better noise bookkeeping.

Our Implementation. Our implementation was based on the NTL C++ library running over GMP, we utilized a machine which consisted of a processing unit of Intel Xeon CPUs running at 2.0 GHz with 18MB cache, and most importantly with 256GB of RAM.²

Memory was our main limiting factor in the implementation. With this machine it took us just under two days to compute a single block AES encryption using an implementation choice which minimizes the amount of memory required; this is roughly two orders of magnitude faster than what could be done with the Gentry-Halevi implementation [14]. The computation was performed on ciphertexts that could hold 864 plaintext slots each; where each slot holds an element of \mathbb{F}_{2^8} . This means that we can compute $\lfloor 864/16 \rfloor = 54$ AES operations in parallel, which gives an amortize time per block of roughly forty minutes. A second (byte-sliced) implementation, requiring more memory, completed an AES operation in around five days; where ciphertexts could hold 720 different \mathbb{F}_{2^8} slots (hence we can evaluate 720 blocks in parallel). This results in an amortized time per block of roughly five minutes.

We note that there are a multitude of optimizations that one can perform on our basic implementation. Most importantly, we believe that by using the “bootstrapping as optimization” technique from BGV [5] we can speedup the AES performance by an additional order of magnitude. Also, there are great gains to be had by making better use of parallelism: Unfortunately, the NTL library (which serves as our underlying software platform) is not thread safe, which severely limits our ability to utilize the multi-core functionality of modern processors (our test machine has 24 cores). We expect that by utilizing many threads we can speed up some of our (higher memory) AES variants by as much as a 16x factor; just by letting each thread compute a different S-box lookup.

Organization. In Section 2 we review the main features of BGV-type cryptosystems [6, 5], and briefly survey the techniques for homomorphic computation on packed ciphertexts from SV and GHS [27, 15].

¹The need for coefficient representation ultimately stems from the fact that the noise in the ciphertexts is small in coefficient representation but not in evaluation representation.

²This machine was BlueCrystal Phase 2; and the authors would like to thank the University of Bristol’s Advanced Computing Research Centre (<https://www.acrc.bris.ac.uk/>) for access to this facility.

Then in Section 3 we describe our “general-purpose” optimizations on a high level, with additional details provided in Appendices A and B. A brief overview of AES and a high-level description and performance numbers is provided in Section 4.

2 Background

2.1 Notations and Mathematical Background

For an integer q we identify the ring $\mathbb{Z}/q\mathbb{Z}$ with the interval $(-q/2, q/2] \cap \mathbb{Z}$, and use $[z]_q$ to denote the reduction of the integer z modulo q into that interval. Our implementation utilizes polynomial rings defined by cyclotomic polynomials, $\mathbb{A} = \mathbb{Z}[X]/\Phi_m(X)$. The ring \mathbb{A} is the ring of integers of the m th cyclotomic number field $\mathbb{Q}(\zeta_m)$. We let $\mathbb{A}_q \stackrel{\text{def}}{=} \mathbb{A}/q\mathbb{A} = \mathbb{Z}[X]/(\Phi_m(X), q)$ for the (possibly composite) integer q , and we identify \mathbb{A}_q with the set of integer polynomials of degree upto $\phi(m) - 1$ reduced modulo q .

Coefficient vs. Evaluation Representation. Let m, q be two integers such that $\mathbb{Z}/q\mathbb{Z}$ contains a primitive m -th root of unity, and denote one such primitive m -th root of unity by $\zeta \in \mathbb{Z}/q\mathbb{Z}$. Recall that the m ’th cyclotomic polynomial splits into linear terms modulo q , $\Phi_m(X) = \prod_{i \in (\mathbb{Z}/m\mathbb{Z})^*} (X - \zeta^i) \pmod{q}$.

We consider two ways of representing an element $a \in \mathbb{A}_q$: Viewing a as a degree- $(\phi(m) - 1)$ polynomial, $a(X) = \sum_{i < \phi(m)} a_i X^i$, the *coefficient representation* of a just lists all the coefficients in order $\mathbf{a} = \langle a_0, a_1, \dots, a_{\phi(m)-1} \rangle \in (\mathbb{Z}/q\mathbb{Z})^{\phi(m)}$. For the other representation we consider the values that the polynomial $a(X)$ assumes on all primitive m -th roots of unity modulo q , $b_i = a(\zeta^i) \pmod{q}$ for $i \in (\mathbb{Z}/m\mathbb{Z})^*$. The b_i ’s in order also yield a vector $\mathbf{b} \in (\mathbb{Z}/q\mathbb{Z})^{\phi(m)}$, which we call the *evaluation representation* of a . Clearly these two representations are related via $\mathbf{b} = V_m \cdot \mathbf{a}$, where V_m is the Vandermonde matrix over the primitive m -th roots of unity modulo q . We remark that for all i we have the equality $(a \pmod{(X - \zeta^i)}) = a(\zeta^i) = b_i$, hence the evaluation representation of a is just a polynomial Chinese-Remaindering representation.

In both representations, an element $a \in \mathbb{A}_q$ is represented by a $\phi(m)$ -vector of integers in $\mathbb{Z}/q\mathbb{Z}$. If q is a composite then each of these integers can itself be represented either using the standard binary encoding of integers or using Chinese-Remaindering relative to the factors of q . We usually use the standard binary encoding for the coefficient representation and Chinese-Remaindering for the evaluation representation. (Hence the latter representation is really a *double CRT* representation, relative to both the polynomial factors of $\Phi_m(X)$ and the integer factors of q .)

2.2 BGV-type Cryptosystems

Our implementation uses a variant of the BGV cryptosystem due to Gentry, Halevi and Smart, specifically the one described in [15, Appendix D] (in the full version). In this cryptosystem both ciphertexts and secret keys are vectors over the polynomial ring \mathbb{A} , and the native plaintext space is the space of binary polynomials \mathbb{A}_2 . (More generally it could be \mathbb{A}_p for some fixed $p \geq 2$, but in our case we will always use \mathbb{A}_2 .)

At any point during the homomorphic evaluation there is some “current integer modulus q ” and “current secret key \mathbf{s} ”, that change from time to time. A ciphertext \mathbf{c} is decrypted using the current secret key \mathbf{s} by taking inner product over \mathbb{A}_q (with q the current modulus) and then reducing the result modulo 2 in *coefficient representation*. Namely, the decryption formula is

$$a \leftarrow \left[\underbrace{[\langle \mathbf{c}, \mathbf{s} \rangle \pmod{\Phi_m(X)}]_q}_{\text{noise}} \right]_2. \quad (1)$$

The polynomial $[\langle \mathbf{c}, \mathbf{s} \rangle \bmod \Phi_m(X)]_q$ is called the “noise” in the ciphertext \mathbf{c} . Informally, \mathbf{c} is a *valid ciphertext* with respect to secret key \mathbf{s} and modulus q if this noise has “sufficiently small norm” relative to q . The meaning of “sufficiently small norm” is whatever is needed to ensure that the noise does not wrap around q when performing homomorphic operations, in our implementation we keep the norm of the noise always below some pre-set bound (which is determined in Appendix C.2).

Following [22, 15], the specific norm that we use to evaluate the magnitude of the noise is the “canonical embedding norm reduced mod q ”, specifically we use the conventions as described in [15, Appendix D] (in the full version). This is useful to get smaller parameters, but for the purpose of presentation the reader can think of the norm as the Euclidean norm of the noise in coefficient representation. More details are given in the Appendices. We refer to the norm of the noise as *the noise magnitude*.

The central feature of BGV-type cryptosystems is that the current secret key and modulus evolve as we apply operations to ciphertexts. We apply five different operations to ciphertexts during homomorphic evaluation. Three of them — addition, multiplication, and automorphism — are “semantic operations” that we use to evolve the plaintext data which is encrypted under those ciphertexts. The other two operations — key-switching and modulus-switching — are used for “maintenance”: These operations do not change the plaintext at all, they only change the current key or modulus (respectively), and they are mainly used to control the complexity of the evaluation. Below we briefly describe each of these five operations on a high level. For the sake of self-containment, we also describe key generation and encryption in Appendix B. More detailed description can be found in [15, Appendix D].

Addition. Homomorphic addition of two ciphertext vectors with respect to the same secret key and modulus q is done just by adding the vectors over \mathbb{A}_q . If the two arguments were encrypting the plaintext polynomials $a_1, a_2 \in \mathbb{A}_2$ then the sum will be an encryption of $a_1 + a_2 \in \mathbb{A}_2$. This operation has no effect on the current modulus or key, and the norm of the noise is at most the sum of norms from the noise in the two arguments.

Multiplication. Homomorphic multiplication is done via tensor product over \mathbb{A}_q . In principle, if the two arguments have dimension n over \mathbb{A}_q then the product ciphertext has dimension n^2 , each entry in the output computed as the product of one entry from the first argument and one entry from the second.³

This operation does not change the current modulus, but it changes the current key: If the two input ciphertexts are valid with respect to the dimension- n secret key vector \mathbf{s} , encrypting the plaintext polynomials $a_1, a_2 \in \mathbb{A}_2$, then the output is valid with respect to the dimension- n^2 secret key \mathbf{s}' which is the tensor product of \mathbf{s} with itself, and it encrypts the polynomial $a_1 \cdot a_2 \in \mathbb{A}_2$. The norm of the noise in the product ciphertext can be bounded in terms of the product of norms of the noise in the two arguments. For our choice of norm function, the norm of the product is no larger than the product of the norms of the two arguments.

Key Switching. The public key of BGV-type cryptosystems includes additional components to enable converting a valid ciphertext with respect to one key into a valid ciphertext encrypting the same plaintext with respect to another key. For example, this is used to convert the product ciphertext which is valid with respect to a high-dimension key back to a ciphertext with respect to the original low-dimension key.

To allow conversion from dimension- n' key \mathbf{s}' to dimension- n key \mathbf{s} (both with respect to the same modulus q), we include in the public key a matrix $W = W[\mathbf{s}' \rightarrow \mathbf{s}]$ over \mathbb{A}_q , where the i 'th column of W is roughly an encryption of the i 'th entry of \mathbf{s}' with respect to \mathbf{s} (and the current modulus). Then given a valid ciphertext \mathbf{c}' with respect to \mathbf{s}' , we roughly compute $\mathbf{c} = W \cdot \mathbf{c}'$ to get a valid ciphertext with respect to \mathbf{s} .

³It was shown in [7] that over polynomial rings this operation can be implemented while increasing the dimension only to $2n - 1$ rather than to n^2 .

In some more detail, the BGV key switching transformation first ensures that the norm of the ciphertext \mathbf{c}' itself is sufficiently low with respect to q . In [5] this was done by working with the binary encoding of \mathbf{c}' , and one of our main optimization in this work is a different method for achieving the same goal (cf. Section 3.1). Then, if the i 'th entry in \mathbf{s}' is $s'_i \in \mathbb{A}$ (with norm smaller than q), then the i 'th column of $W[\mathbf{s}' \rightarrow \mathbf{s}]$ is an n -vector \mathbf{w}_i such that $[\langle \mathbf{w}_i, \mathbf{s} \rangle \bmod \Phi_m(X)]_q = 2e_i + s'_i$ for a low-norm polynomial $e_i \in \mathbb{A}$. Denoting $\mathbf{e} = (e_1, \dots, e_{n'})$, this means that we have $\mathbf{s}W = \mathbf{s}' + 2\mathbf{e}$ over \mathbb{A}_q . For any ciphertext vector \mathbf{c}' , setting $\mathbf{c} = W \cdot \mathbf{c}' \in \mathbb{A}_q$ we get the equation

$$[\langle \mathbf{c}, \mathbf{s} \rangle \bmod \Phi_m(X)]_q = [\mathbf{s}W\mathbf{c}' \bmod \Phi_m(X)]_q = [\langle \mathbf{c}', \mathbf{s}' \rangle + 2\langle \mathbf{c}', \mathbf{e} \rangle \bmod \Phi_m(X)]_q$$

Since \mathbf{c}' , \mathbf{e} , and $[\langle \mathbf{c}', \mathbf{s}' \rangle \bmod \Phi_m(X)]_q$ all have low norm relative to q , then the addition on the right-hand side does not cause a wrap around q , hence we get $[[\langle \mathbf{c}, \mathbf{s} \rangle \bmod \Phi_m(X)]_q]_2 = [[\langle \mathbf{c}', \mathbf{s}' \rangle \bmod \Phi_m(X)]_q]_2$, as needed. The key-switching operation changes the current secret key from \mathbf{s}' to \mathbf{s} , and does not change the current modulus. The norm of the noise is increased by at most an additive factor of $2\|\langle \mathbf{c}', \mathbf{e} \rangle\|$.

Modulus Switching. The modulus switching operation is intended to reduce the norm of the noise, to compensate for the noise increase that results from all the other operations. To convert a ciphertext \mathbf{c} with respect to secret key \mathbf{s} and modulus q into a ciphertext \mathbf{c}' encrypting the same thing with respect to the same secret key but modulus q' , we roughly just scale \mathbf{c} by a factor q'/q (thus getting a fractional ciphertext), then round appropriately to get back an integer ciphertext. Specifically \mathbf{c}' is a ciphertext vector satisfying (a) $\mathbf{c}' = \mathbf{c} \pmod{2}$, and (b) the “rounding error term” $\tau \stackrel{\text{def}}{=} \mathbf{c}' - (q'/q)\mathbf{c}$ has low norm. Converting \mathbf{c} to \mathbf{c}' is easy in coefficient representation, and one of our optimizations is a method for doing the same in evaluation representation (cf. Section 3.2) This operation leaves the current key \mathbf{s} unchanged, changes the current modulus from q to q' , and the norm of the noise is changed as $\|n'\| \leq (q'/q)\|n\| + \|\tau \cdot \mathbf{s}\|$. Note that if the key \mathbf{s} has low norm and q' is sufficiently smaller than q , then the noise magnitude decreases by this operation.

A BGV-type cryptosystem has a chain of moduli, $q_0 < q_1 \cdots < q_{L-1}$, where fresh ciphertexts are with respect to the largest modulus q_{L-1} . During homomorphic evaluation every time the (estimated) noise grows too large we apply modulus switching from q_i to q_{i-1} in order to decrease it back. Eventually we get ciphertexts with respect to the smallest modulus q_0 , and we cannot compute on them anymore (except by using bootstrapping).

Automorphisms. In addition to adding and multiplying polynomials, another useful operation is converting the polynomial $a(X) \in \mathbb{A}$ to $a^{(i)}(X) \stackrel{\text{def}}{=} a(X^i) \bmod \Phi_m(X)$. Denoting by κ_i the transformation $\kappa_i : a \mapsto a^{(i)}$, it is a standard fact that the set of transformations $\{\kappa_i : i \in (\mathbb{Z}/m\mathbb{Z})^*\}$ forms a group under composition (which is the Galois group $\mathcal{Gal}(\mathbb{Q}(\zeta_m)/\mathbb{Q})$), and this group is isomorphic to $(\mathbb{Z}/m\mathbb{Z})^*$. In [5, 15] it was shown that applying the transformations κ_i to the plaintext polynomials is very useful, some more examples of its use can be found in our Section 4.

Denoting by $\mathbf{c}^{(i)}$, $\mathbf{s}^{(i)}$ the vector obtained by applying κ_i to each entry in \mathbf{c} , \mathbf{s} , respectively, it was shown in [5, 15] that if \mathbf{s} is a valid ciphertext encrypting a with respect to key \mathbf{s} and modulus q , then $\mathbf{c}^{(i)}$ is a valid ciphertext encrypting $a^{(i)}$ with respect to key $\mathbf{s}^{(i)}$ and the same modulus q . Moreover the norm of noise remains the same under this operation. We remark that we can apply key-switching to $\mathbf{c}^{(i)}$ in order to get an encryption of $a^{(i)}$ with respect to the original key \mathbf{s} .

2.3 Computing on Packed Ciphertexts

Smart and Vercauteren observed [26, 27] that the plaintext space \mathbb{A}_2 can be viewed as a vector of “plaintext slots”, by an application the polynomial Chinese Remainder Theorem. Specifically, if the ring polynomial $\Phi_m(X)$ factors modulo 2 into a product of irreducible factors $\Phi_m(X) = \prod_{j=0}^{\ell-1} F_j(X) \pmod{2}$, then a plaintext polynomial $a(X) \in \mathbb{A}_2$ can be viewed as encoding ℓ different small polynomials, $a_j = a \bmod F_j$. Just like for integer Chinese Remaindering, addition and multiplication in \mathbb{A}_2 correspond to element-wise addition and multiplication of the vectors of slots.

The effect of the automorphisms is a little more involved. When i is a power of two then the transformations $\kappa_i : a \mapsto a^{(i)}$ is just applied to each slot separately. When i is not a power of two the transformation κ_i has the effect of roughly shifting the values between the different slots. For example, for some parameters we could get a cyclic shift of the vector of slots: If a encodes the vector $(a_0, a_1, \dots, a_{\ell-1})$, then $\kappa_i(a)$ (for some i) could encode the vector $(a_{\ell-1}, a_0, \dots, a_{\ell-2})$. This was used in [15] to devise efficient procedures for applying arbitrary permutations to the plaintext slots.

We note that the values in the plaintext slots are not just bits, rather they are polynomials modulo the irreducible F_j ’s, so they can be used to represent elements in extension fields $\text{GF}(2^d)$. In particular, in some of our AES implementations we used the plaintext slots to hold elements of $\text{GF}(2^8)$, and encrypt one byte of the AES state in each slot. Then we can use an adaption of the techniques from [15] to permute the slots when performing the AES row-shift and column-mix.

3 General-Purpose Optimizations

Below we summarize our optimizations that are not tied directly to the AES circuit and can be used also in homomorphic evaluation of other circuits. Underlying many of these optimizations is our choice of keeping ciphertext and key-switching matrices in evaluation (double-CRT) representation. Our chain of moduli is defined via a set of primes of roughly the same size, p_0, \dots, p_{L-1} , all chosen such that $\mathbb{Z}/p_i\mathbb{Z}$ has a m ’th roots of unity. (In other words, $m|p_i - 1$ for all i .) For $i = 0, \dots, L - 1$ we then define our i ’th modulus as $q_i = \prod_{j=0}^i p_j$. The primes p_0 and p_{L-1} are special (p_0 is chosen to ensure decryption works, and p_{L-1} is chosen to control noise immediately after encryption), however all other primes p_i are of size $2^{17} \leq p_i \leq 2^{20}$ if $L < 100$, see Appendix C.

In the t -th level of the scheme we have ciphertexts consisting of elements in \mathbb{A}_{q_t} (i.e., polynomials modulo $(\Phi_m(X), q_t)$). We represent an element $c \in \mathbb{A}_{q_t}$ by a $\phi(m) \times (t + 1)$ “matrix” of its evaluations at the primitive m -th roots of unity modulo the primes p_0, \dots, p_t . Computing this representation from the coefficient representation of c involves reducing c modulo the p_i ’s and then $t + 1$ invocations of the FFT algorithm, modulo each of the p_i (picking only the FFT coefficients corresponding to $(\mathbb{Z}/m\mathbb{Z})^*$). To convert back to coefficient representation we invoke the inverse FFT algorithm $t + 1$ times, each time padding the $\phi(m)$ -vector of evaluation point with $m - \phi(m)$ zeros (for the evaluations at the non-primitive roots of unity). This yields the coefficients of $t + 1$ polynomials modulo $(X^m - 1, p_i)$ for $i = 0, \dots, t$, we then reduce each of these polynomials modulo $(\Phi_m(X), p_i)$ and apply Chinese Remainder interpolation. We stress that we try to perform these transformations as rarely as we can.

3.1 A New Variant of Key Switching

As described in Section 2, the key-switching transformation introduces an additive factor of $2 \langle \mathbf{c}', \mathbf{e} \rangle$ in the noise, where \mathbf{c}' is the input ciphertext and \mathbf{e} is the noise component in the key-switching matrix. To keep the noise magnitude below the modulus q , it seems that we need to ensure that the ciphertext \mathbf{c}'

itself has low norm. In BGV [5] this was done by representing \mathbf{c}' as a fixed linear combination of small vectors, i.e. $\mathbf{c}' = \sum_i 2^i \mathbf{c}'_i$ with \mathbf{c}'_i the vector of i 'th bits in \mathbf{c}' . Considering the high-dimension ciphertext $\mathbf{c}^* = (\mathbf{c}'_0 | \mathbf{c}'_1 | \mathbf{c}'_2 | \dots)$ and secret key $\mathbf{s}^* = (\mathbf{s}' | 2\mathbf{s}' | 4\mathbf{s}' | \dots)$, we note that we have $\langle \mathbf{c}^*, \mathbf{s}^* \rangle = \langle \mathbf{c}', \mathbf{s}' \rangle$, and \mathbf{c}^* has low norm (since it consists of 0-1 polynomials). BGV therefore included in the public key the matrix $W = W[\mathbf{s}^* \rightarrow \mathbf{s}]$ (rather than $W[\mathbf{s}' \rightarrow \mathbf{s}]$), and had the key-switching transformation computes \mathbf{c}^* from \mathbf{c}' and sets $\mathbf{c} = W \cdot \mathbf{c}^*$.

When implementing key-switching, there are two drawbacks to the above approach. First, this increases the dimension (and hence the size) of the key switching matrix. This drawback is fatal when evaluating deep circuits, since having enough memory to keep the key-switching matrices turns out to be the limiting factor in our ability to evaluate these deep circuits. In addition, for this key-switching we must first convert \mathbf{c}' to coefficient representation (in order to compute the \mathbf{c}'_i 's), then convert each of the \mathbf{c}'_i 's back to evaluation representation before multiplying by the key-switching matrix. In level t of the circuit, this seem to require $\Omega(t \log q_t)$ FFTs.

In this work we propose a different variant: Rather than manipulating \mathbf{c}' to decrease its norm, we instead temporarily increase the modulus q . We recall that for a valid ciphertext \mathbf{c}' , encrypting plaintext a with respect to \mathbf{s}' and q , we have the equality $\langle \mathbf{c}', \mathbf{s}' \rangle = 2e' + a$ over A_q , for a low-norm polynomial e' . This equality, we note, implies that for every odd integer p we have the equality $\langle \mathbf{c}', p\mathbf{s}' \rangle = 2e'' + a$, *holding over A_{pq}* , for the “low-norm” polynomial e'' (namely $e'' = p \cdot e' + \frac{p-1}{2}a$). Clearly, when considered relative to secret key $p\mathbf{s}$ and modulus pq , the noise in \mathbf{c}' is p times larger than it was relative to \mathbf{s} and q . However, since the modulus is also p times larger, we maintain that the noise has norm sufficiently smaller than the modulus. In other words, \mathbf{c}' is still a valid ciphertext that encrypts the same plaintext a with respect to secret key $p\mathbf{s}$ and modulus pq . By taking p large enough, we can ensure that the norm of \mathbf{c}' (which is independent of p) is sufficiently small relative to the modulus pq .

We therefore include in the public key a matrix $W = W[p\mathbf{s}' \rightarrow \mathbf{s}]$ modulo pq for a large enough odd integer p . (Specifically we need $p \approx q\sqrt{m}$.) Given a ciphertext \mathbf{c}' , valid with respect to \mathbf{s} and q , we apply the key-switching transformation simply by setting $\mathbf{c} = W \cdot \mathbf{c}'$ over \mathbb{A}_{pq} . The additive noise term $\langle \mathbf{c}', \mathbf{e} \rangle$ that we get is now small enough relative to our large modulus pq , thus the resulting ciphertext \mathbf{c} is valid with respect to \mathbf{s} and pq . We can now switch the modulus back to q (using our modulus switching routine), hence getting a valid ciphertext with respect to \mathbf{s} and q .

We note that even though we no longer break \mathbf{c}' into its binary encoding, it seems that we still need to recover it in coefficient representation in order to compute the evaluations of $\mathbf{c}' \bmod p$. However, since we do not increase the dimension of the ciphertext vector, this procedure requires only $O(t)$ FFTs in level t (vs. $O(t \log q_t) = O(t^2)$ for the original BGV variant). Also, the size of the key-switching matrix is reduced by roughly the same factor of $\log q_t$.

Our new variant comes with a price tag, however: We use key-switching matrices relative to a larger modulus, but still need the noise term in this matrix to be small. This means that the LWE problem underlying this key-switching matrix has larger ratio of modulus/noise, implying that we need a larger dimension to get the same level of security than with the original BGV variant. In fact, since our modulus is more than squared (from q to pq with $p > q$), the dimension is increased by more than a factor of two. This translates to more than doubling of the key-switching matrix, partly negating the size and running time advantage that we get from this variant.

We comment that a hybrid of the two approaches could also be used: we can decrease the norm of \mathbf{c}' only somewhat by breaking it into digits (as opposed to binary bits as in [5]), and then increase the modulus somewhat until it is large enough relative to the smaller norm of \mathbf{c}' . We speculate that the optimal setting in terms of runtime is found around $p \approx \sqrt{q}$, but so far did not try to explore this tradeoff.

3.2 Modulus Switching in Evaluation Representation

Given an element $c \in \mathbb{A}_{q_t}$ in evaluation (double-CRT) representation relative to $q_t = \prod_{j=0}^t p_j$, we want to modulus-switch to q_{t-1} – i.e., scale down by a factor of p_t ; we call this operation $\text{Scale}(c, q_t, q_{t-1})$. The output should be $c' \in \mathbb{A}$, represented via the same double-CRT format (with respect to p_0, \dots, p_{t-1}), such that (a) $c' \equiv c \pmod{2}$, and (b) the “rounding error term” $\tau = c' - (c/p_t)$ has a very low norm. As p_t is odd, we can equivalently require that the element $c^\dagger \stackrel{\text{def}}{=} p_t \cdot c'$ satisfy

- (i) c^\dagger is divisible by p_t ,
- (ii) $c^\dagger \equiv c \pmod{2}$, and
- (iii) $c^\dagger - c$ (which is equal to $p_t \cdot \tau$) has low norm.

Rather than computing c' directly, we will first compute c^\dagger and then set $c' \leftarrow c^\dagger/p_t$. Observe that once we compute c^\dagger in double-CRT format, it is easy to output also c' in double-CRT format: given the evaluations for c^\dagger modulo p_j ($j < t$), simply multiply them by $p_t^{-1} \pmod{p_j}$. The algorithm to output c^\dagger in double-CRT format is as follows:

1. Set \bar{c} to be the coefficient representation of $c \pmod{p_t}$. (Computing this requires a single “small FFT” modulo the prime p_t .)
2. Add or subtract p_t from every odd coefficient of \bar{c} , thus obtaining a polynomial δ with coefficients in $(-p_t, p_t]$ such that $\delta \equiv \bar{c} \equiv c \pmod{p_t}$ and $\delta \equiv 0 \pmod{2}$.
3. Set $c^\dagger = c - \delta$, and output it in double-CRT representation.

Since we already have c in double-CRT representation, we only need the double-CRT representation of δ , which requires t more “small FFTs” modulo the p_j ’s.

As all the coefficients of c^\dagger are within p_t of those of c , the “rounding error term” $\tau = (c^\dagger - c)/p_t$ has coefficients of magnitude at most one, hence it has low norm.

The procedure above uses $t + 1$ small FFTs in total. This should be compared to the naive method of just converting everything to coefficient representation modulo the primes ($t + 1$ FFTs), CRT-interpolating the coefficients, dividing and rounding appropriately the large integers (of size $\approx q_t$), CRT-decomposing the coefficients, and then converting back to evaluation representation ($t + 1$ more FFTs). The above approach makes explicit use of the fact that we are working in a plaintext space modulo 2; in Appendix D we present a technique which works when the plaintext space is defined modulo a larger modulus.

3.3 Dynamic Noise Management

As described in the literature, BGV-type cryptosystems tacitly assume that each homomorphic operation is followed a modulus switch to reduce the noise magnitude. In our implementation, however, we attach to each ciphertext an estimate of the noise magnitude in that ciphertext, and use these estimates to decide dynamically when a modulus switch must be performed.

Each modulus switch consumes a level, and hence a goal is to reduce, over a computation, the number of levels consumed. By paying particular attention to the parameters of the scheme, and by carefully analyzing how various operations affect the noise, we are able to control the noise much more carefully than in prior work. In particular, we note that modulus-switching is really only necessary just prior to multiplication (when the noise magnitude is about to get squared), in other times it is acceptable to keep the ciphertexts at a higher level (with higher noise).

3.4 Randomized Multiplication by Constants

Our implementation of the AES round function uses just a few multiplication operations (only seven per byte!), but it requires a relatively large number of multiplications of encrypted bytes by constants. Hence it becomes important to try and squeeze down the increase in noise when multiplying by a constant. To that end, we encode a constant polynomial in \mathbb{A}_2 as a polynomial with coefficients in $\{-1, 0, 1\}$ rather than in $\{0, 1\}$. Namely, we have a procedure $\text{Randomize}(\alpha)$ that takes a polynomial $\alpha \in \mathbb{A}_2$ and replaces each non-zero coefficients with a coefficients chosen uniformly from $\{-1, 1\}$. By Chernoff bound, we expect that for α with h nonzero coefficients, the canonical embedding norm of $\text{Randomize}(\alpha)$ to be bounded by $O(\sqrt{h})$ with high probability (assuming that h is large enough for the bound to kick in). This yields a better bound on the noise increase than the trivial bound of h that we would get if we just multiply by α itself. (In Appendix A.5 we present a heuristic argument that we use to bound the noise, which yields the same asymptotic bounds but slightly better constants.)

4 Homomorphic Evaluation of AES

Next we describe our homomorphic implementation of AES-128. We implemented three distinct implementation possibilities; we first describe the “packed implementation”, in which the entire AES state is packed in just one ciphertext. Two other implementations (of byte-slice and bit-slice AES) are described later in Section 4.2. The “packed” implementation uses the least amount of memory (which turns out to be the main constraint in our implementation), and also the fastest running time for a single evaluation. The other implementation choices allow more SIMD parallelism, on the other hand, so they can give better amortized running time when evaluating AES on many blocks in parallel.

A Brief Overview of AES. The AES-128 cipher consists of ten applications of the same keyed round function (with different round keys). The round function operates on a 4×4 matrix of bytes, which are sometimes considered as element of \mathbb{F}_{2^8} . The basic operations that are performed during the round function are AddKey, SubBytes, ShiftRows, MixColumns. The AddKey is simply an XOR operation of the current state with 16 bytes of key; the SubBytes operation consists of an inversion in the field \mathbb{F}_{2^8} followed by a fixed \mathbb{F}_2 -linear map on the bits of the element (relative to a fixed polynomial representation of \mathbb{F}_{2^8}); the ShiftRows rotates the entries in the row i of the 4×4 matrix by $i - 1$ places to the left; finally the MixColumns operations pre-multiplies the state matrix by a fixed 4×4 matrix.

Our Packed Representation of the AES state. For our implementation we chose the native plaintext space of our homomorphic encryption so as to support operations on the finite field \mathbb{F}_{2^8} . To this end we choose our ring polynomial as $\Phi_m(X)$ that factors modulo 2 into degree- d irreducible polynomials such that $8|d$. (In other words, the smallest integer d such that $m|(2^d - 1)$ is divisible by 8.) This means that our plaintext slots can hold elements of \mathbb{F}_{2^d} , and in particular we can use them to hold elements of \mathbb{F}_{2^8} which is a sub-field of \mathbb{F}_{2^d} . Since we have $\ell = \phi(m)/d$ plaintext slots in each ciphertext, we can represent upto $\lfloor \ell/16 \rfloor$ complete AES state matrices per ciphertext.

Moreover, we choose our parameter m so that there exists an element $g \in \mathbb{Z}_m^*$ that has order 16 in both \mathbb{Z}_m^* and the quotient group $\mathbb{Z}_m^*/\langle 2 \rangle$. This condition means that if we put 16 plaintext bytes in slots t, tg, tg^2, tg^3, \dots (for some $t \in \mathbb{Z}_m^*$), then the conjugation operation $X \mapsto X^g$ implements a cyclic right shift over these sixteen plaintext bytes.

In the computation of the AES round function we use several constants. Some constants are used in the S-box lookup phase to implement the AES bit-affine transformation, these are denoted γ and γ_{2^j} for $j = 0, \dots, 7$. In the row-shift/col-mix part we use a constant C_{slet} that has 1 in slots corresponding to $t \cdot g^i$ for $i = 0, 4, 8, 12$, and 0 in all the other slots of the form $t \cdot g^i$. (Here slot t is where we put the first AES byte.) We also use 'X' to denote the constant that has the element X in all the slots.

4.1 Homomorphic Evaluation of the Basic Operations

We now examine each AES operation in turn, and describe how it is implemented homomorphically. For each operation we denote the plaintext polynomial underlying a given input ciphertext c by a , and the corresponding content of the ℓ plaintext slots are denoted as an ℓ -vector $(\alpha_i)_{i=1}^\ell$, with each $\alpha_i \in \mathbb{F}_{2^8}$.

4.1.1 AddKey and SubBytes

The AddKey is just a simple addition of ciphertexts, which yields a 4×4 matrix of bytes in the input to the SubBytes operation. We place these 16 bytes in plaintext slots tg^i for $i = 0, 1, \dots, 15$, using column-ordering to decide which byte goes in what slot, namely we have

$$a \approx [\alpha_{00} \alpha_{10} \alpha_{20} \alpha_{30} \alpha_{01} \alpha_{11} \alpha_{21} \alpha_{31} \alpha_{02} \alpha_{12} \alpha_{22} \alpha_{32} \alpha_{03} \alpha_{13} \alpha_{23} \alpha_{33}],$$

encrypting the input plaintext matrix

$$A = (\alpha_{ij})_{i,j} = \begin{pmatrix} \alpha_{00} & \alpha_{01} & \alpha_{02} & \alpha_{03} \\ \alpha_{10} & \alpha_{11} & \alpha_{12} & \alpha_{13} \\ \alpha_{20} & \alpha_{21} & \alpha_{22} & \alpha_{23} \\ \alpha_{30} & \alpha_{31} & \alpha_{32} & \alpha_{33} \end{pmatrix}.$$

During S-box lookup, each plaintext byte α_{ij} should be replaced by $\beta_{ij} = S(\alpha_{ij})$, where $S(\cdot)$ is a fixed permutation on the bytes. Specifically, $S(x)$ is obtained by first computing $y = x^{-1}$ in \mathbb{F}_{2^8} (with 0 mapped to 0), then applying a bitwise affine transformation $z = T(y)$ where elements in \mathbb{F}_{2^8} are treated as bit strings with representation polynomial $G(X) = x^8 + x^4 + x^3 + x + 1$.

We implement \mathbb{F}_{2^8} inversion followed by the \mathbb{F}_2 affine transformation using the Frobenius automorphisms, $X \rightarrow X^{2^j}$. Recall that for a power of two $k = 2^j$, the transformation $\kappa_k(a(X)) = (a(X^k) \bmod \Phi_m(X))$ is applied separately to each slot, hence we can use it to transform the vector $(\alpha_i)_{i=1}^\ell$ into $(\alpha_i^k)_{i=1}^\ell$. We note that applying the Frobenius automorphisms to ciphertexts has almost no influence on the noise magnitude, and hence it does not consume any levels.⁴

Inversion over \mathbb{F}_{2^8} is done using essentially the same procedure as Algorithm 2 from [25] for computing $\beta = \alpha^{-1} = \alpha^{2^{54}}$. This procedure takes only three Frobenius automorphisms and four multiplications, arranged in a depth-3 circuit (see details below.) To apply the AES \mathbb{F}_2 affine transformation, we use the fact that any \mathbb{F}_2 affine transformation can be computed as a \mathbb{F}_{2^8} affine transformation over the conjugates. Thus there are constants $\gamma_0, \gamma_1, \dots, \gamma_7, \delta \in \mathbb{F}_{2^8}$ such that the AES affine transformation $T_{\text{AES}}(\cdot)$ can be expressed as $T_{\text{AES}}(\beta) = \delta + \sum_{j=0}^7 \gamma_j \cdot \beta^{2^j}$ over \mathbb{F}_{2^8} . We therefore again apply the Frobenius automorphisms to compute eight ciphertexts encrypting the polynomials $\kappa_k(b)$ for $k = 1, 2, 4, \dots, 128$, and take the appropriate linear combination (with coefficients the γ_j 's) to get an encryption of the vector $(T_{\text{AES}}(\alpha_i^{-1}))_{i=1}^\ell$. For our parameters, a multiplication-by-constant operation consumes roughly half a level in terms of added noise.

⁴It does increase the noise magnitude somewhat, because we need to do key switching after these automorphisms. But this is only a small influence, and we will ignore it here.

One subtle implementation detail to note here, is that although our plaintext slots all hold elements of the same field \mathbb{F}_{2^8} , they hold these elements with respect to different polynomial encodings. The AES affine transformation, on the other hand, is defined with respect to one particular fixed polynomial encoding. This means that we must implement in the i 'th slot not the affine transformation $T_{\text{AES}}(\cdot)$ itself but rather the projection of this transformation onto the appropriate polynomial encoding: When we take the affine transformation of the eight ciphertexts encrypting $b_j = \kappa_{2^j}(b)$, we therefore multiply the encryption of b_j not by a constant that has γ_j in all the slots, but rather by a constant that has in slot i the projection of γ_j to the polynomial encoding of slot i .

Below we provide a pseudo-code description of our S-box lookup implementation, together with an approximation of the levels that are consumed by these operations. (These approximations are somewhat under-estimates, however.)

Input: ciphertext c	Level
t	
// Compute $c_{254} = c^{-1}$	
1. $c_2 \leftarrow c \ggg 2$	t // Frobenius $X \mapsto X^2$
2. $c_3 \leftarrow c \times c_2$	$t + 1$ // Multiplication
3. $c_{12} \leftarrow c_3 \ggg 4$	$t + 1$ // Frobenius $X \mapsto X^4$
4. $c_{14} \leftarrow c_{12} \times c_2$	$t + 2$ // Multiplication
5. $c_{15} \leftarrow c_{12} \times c_3$	$t + 2$ // Multiplication
6. $c_{240} \leftarrow c_{15} \ggg 16$	$t + 2$ // Frobenius $X \mapsto X^{16}$
7. $c_{254} \leftarrow c_{240} \times c_{14}$	$t + 3$ // Multiplication
// Affine transformation over \mathbb{F}_2	
8. $c'_{2^j} \leftarrow c_{254} \ggg 2^j$ for $j = 0, 1, 2, \dots, 7$	$t + 3$ // Frobenius $X \mapsto X^{2^j}$
9. $c'' \leftarrow \gamma + \sum_{j=0}^7 \gamma_j \times c'_{2^j}$	$t + 3.5$ // Linear combination over \mathbb{F}_{2^8}

4.1.2 ShiftRows and MixColumns

As commonly done, we interleave the ShiftRows/MixColumns operations, viewing both as a single linear transformation over vectors from $(\mathbb{F}_{2^8})^{16}$. As mentioned above, by a careful choice of the parameter m and the placement of the AES state bytes in our plaintext slots, we can implement a rotation-by- i of the rows of the AES matrix as a single automorphism operations $X \mapsto X^{g^i}$ (for some element $g \in (\mathbb{Z}/m\mathbb{Z})^*$). Given the ciphertext c'' after the SubBytes step, we use these operations (in conjunction with ℓ -SELECT operations, as described in [15]) to compute four ciphertexts corresponding to the appropriate permutations of the 16 bytes (in each of the $\ell/16$ different input blocks). These four ciphertexts are combined via a linear operation (with coefficients 1, X , and $(1 + X)$) to obtain the final result of this round function. Below is a pseudo-code of this implementation and an approximation for the levels that it consumes (starting from $t = 3.5$). We note that the permutations are implemented using automorphisms and multiplication by constant, thus we expect them to consume roughly 1/2 level.

Input: ciphertext c''	Level
$t + 3.5$	
10. $c_j^* \leftarrow \pi_j(c'')$ for $j = 1, 2, 3, 4$	$t + 4.0$ // Permutations
11. Output $X \cdot c_1^* + (X + 1) \cdot c_2^* + c_3^* + c_4^*$	$t + 4.5$ // Linear combination

4.1.3 The Cost of One Round Function

The above description yields an estimate of 5 levels for implementing one round function. This is however, an underestimate. The actual number of levels depends on details such as how sparse the scalars are with respect to the embedding via Φ_m in a given parameter set, as well as the accumulation of noise with respect to additions, Frobenius operations etc. Running over many different parameter sets we find the average number of levels per round for this method varies between 5.0 and 6.0.

We mention that the byte-slice and bit-slice implementations, given in Section 4.2 below, can consume less levels per round function, since they do not need to permute slots inside a single ciphertext. Specifically, for our byte-sliced implementation, we only need 4.5-5.0 levels per round on average. However, since we need to manipulate many more ciphertexts, the implementation takes much more time per evaluation and requires much more memory. On the other hand it offers wider parallelism, so yields better amortized time per block. Our bit-sliced implementation should theoretical consume the least number of levels (by purely counting multiplication gates), but the noise introduced by additions means the average number of levels consumed per round varies from 5.0 upto 10.0.

4.2 Byte- and Bit-Slice Implementations

In the byte sliced implementation we use sixteen distinct ciphertexts to represent a single state matrix. (But since each ciphertext can hold ℓ plaintext slots, then these 16 ciphertexts can hold the state of ℓ different AES blocks). In this representation there is no interaction between the slots, thus we operate with pure ℓ -fold SIMD operations. The AddKey and SubBytes steps are exactly as above (except applied to 16 ciphertexts rather than a single one). The permutations in the ShiftRows/MixColumns step are now “for free”, but the scalar multiplication in MixColumns still consumes another level in the modulus chain.

Using the same estimates as above, we expect the number of levels per round to be roughly four (as opposed to the 4.5 of the packed implementation). In practice, again over many parameter sets, we find the average number of levels consumed per round is between 4.5 and 5.0.

For the bit sliced implementation we represent the entire round function as a binary circuit, and we use 128 distinct ciphertexts (one per bit of the state matrix). However each set of 128 ciphertexts is able to represent a total of ℓ distinct blocks. The main issue here is how to create a circuit for the round function which is as shallow, in terms of number of multiplication gates, as possible. Again the main issue is the SubBytes operation as all operations are essentially linear. To implement the SubBytes we used the “depth-16” circuit of Boyar and Peralta [3], which consumes four levels. The rest of the round function can be represented as a set of bit-additions. Thus, implementing this method means that we consumes a minimum of four levels on computing an entire round function. However, the extensive additions within the Boyar–Peralta circuit mean that we actually end up consuming a lot more. On average this translates into actually consuming between 5.0 and 10.0 levels per round.

4.3 Performance Details

As remarked in the introduction, we implemented the above variant of evaluating AES homomorphically on a very large memory machine; namely a machine with 256 GB of RAM. Firstly parameters were selected, as in Appendix C, to cope with 60 levels of computation, and a public/private key pair was generated; along with the key-switching data for multiplication operations and conjugation with-respect-to the Galois group.

As input to the actual computation was an AES plaintext block and the eleven round keys; each of which was encrypted using our homomorphic encryption scheme. Thus the input consisted of eleven packed

ciphertexts. Producing the encrypted key schedule took around half an hour. To evaluate the entire ten rounds of AES took just over 36 hours; however each of our ciphertexts could hold 864 plaintext slots of elements in \mathbb{F}_{2^8} , thus we could have processed 54 such AES blocks in this time period. This would result in a throughput of around forty minutes per AES block.

We note that as the algorithm progressed the operations became faster. The first round of the AES function took 7 hours, whereas the penultimate round took 2 hours and the last round took 30 minutes. Recall, the last AES round is somewhat simpler as it does not involve a MixColumns operation.

Whilst our other two implementation choices (given in Section 4.2 below) may seem to yield better amortized per-block timing, the increase in memory requirements and data actually makes them less attractive when encrypting a single block. For example just encrypting the key schedule in the Byte-Sliced variant takes just under 5 hours (with 50 levels), with an entire encryption taking 65 hours (12 hours for the first round, with between 4 and 5 hours for both the penultimate and final rounds). This however equates to an amortized time of just over five minutes per block.

The Bit-Sliced variant requires over 150 hours to just encrypt the key schedule (with 60 levels), and evaluating a single round takes so long that our program is timed out before even a single round is evaluated.

Acknowledgments

We thank Jean-Sebastien Coron for pointing out to us the efficient implementation from [25] of the AES S-box lookup.

References

- [1] Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In *CRYPTO*, volume 5677 of *Lecture Notes in Computer Science*, pages 595–618. Springer, 2009.
- [2] Sanjeev Arora and Rong Ge. New algorithms for learning in the presence of errors. In *ICALP*, volume 6755 of *Lecture Notes in Computer Science*, pages 403–415. Springer, 2011.
- [3] Joan Boyar and René Peralta. A depth-16 circuit for the AES S-box. Manuscript, <http://eprint.iacr.org/2011/332>, 2011.
- [4] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. Manuscript, <http://eprint.iacr.org/2012/078>, 2012.
- [5] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. Fully homomorphic encryption without bootstrapping. In *Innovations in Theoretical Computer Science (ITCS'12)*, 2012. Available at <http://eprint.iacr.org/2011/277>.
- [6] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In *FOCS'11*. IEEE Computer Society, 2011.
- [7] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-LWE and security for key dependent messages. In *Advances in Cryptology - CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 505–524. Springer, 2011.

- [8] Jean-Sébastien Coron, Avradip Mandal, David Naccache, and Mehdi Tibouchi. Fully homomorphic encryption over the integers with shorter public keys. In *Advances in Cryptology - CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 487–504. Springer, 2011.
- [9] Jean-Sébastien Coron, David Naccache, and Mehdi Tibouchi. Public key compression and modulus switching for fully homomorphic encryption over the integers. In *Advances in Cryptology - EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 446–464. Springer, 2012.
- [10] Ivan Damgård and Marcel Keller. Secure multiparty aes. In *Proc. of Financial Cryptography 2010*, volume 6052 of *LNCS*, pages 367–374, 2010.
- [11] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. Manuscript, 2011.
- [12] Nicolas Gama and Phong Q. Nguyen. Predicting lattice reduction. In *EUROCRYPT*, volume 4965 of *Lecture Notes in Computer Science*, pages 31–51. Springer, 2008.
- [13] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *STOC*, pages 169–178. ACM, 2009.
- [14] Craig Gentry and Shai Halevi. Implementing gentry’s fully-homomorphic encryption scheme. In *EUROCRYPT*, volume 6632 of *Lecture Notes in Computer Science*, pages 129–148. Springer, 2011.
- [15] Craig Gentry, Shai Halevi, and Nigel Smart. Fully homomorphic encryption with polylog overhead. In *EUROCRYPT*, volume 7237 of *Lecture Notes in Computer Science*, pages 465–482. Springer, 2012. Full version at <http://eprint.iacr.org/2011/566>.
- [16] Shafi Goldwasser, Yael Tauman Kalai, Chris Peikert, and Vinod Vaikuntanathan. Robustness of the learning with errors assumption. In *Innovations in Computer Science - ICS '10*, pages 230–240. Tsinghua University Press, 2010.
- [17] Yan Huang, David Evans, Jonathan Katz, and Lior Malka. Faster secure two-party computation using garbled circuits. In *USENIX Security Symposium*, 2011.
- [18] C. Orlandi J.B. Nielsen, P.S. Nordholt and S. Sheshank. A new approach to practical active-secure two-party computation. Manuscript, 2011.
- [19] Kristin Lauter, Michael Naehrig, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? In *CCSW*, pages 113–124. ACM, 2011.
- [20] Richard Lindner and Chris Peikert. Better key sizes (and attacks) for lwe-based encryption. In *CT-RSA*, volume 6558 of *Lecture Notes in Computer Science*, pages 319–339. Springer, 2011.
- [21] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *STOC*. ACM, 2012.
- [22] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *EUROCRYPT*, volume 6110 of *Lecture Notes in Computer Science*, pages 1–23, 2010.
- [23] Daniele Micciancio and Oded Regev. *Lattice-based cryptography*, pages 147–192. Springer, 2009.

- [24] Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Steven C. Williams. Secure two-party computation is practical. In *Proc. ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 250–267, 2009.
- [25] Matthieu Rivain and Emmanuel Prouff. Provably secure higher-order masking of AES. In *CHES*, volume 6225 of *Lecture Notes in Computer Science*, pages 413–427. Springer, 2010.
- [26] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In *Public Key Cryptography - PKC'10*, volume 6056 of *Lecture Notes in Computer Science*, pages 420–443. Springer, 2010.
- [27] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic SIMD operations. Manuscript at <http://eprint.iacr.org/2011/133>, 2011.

A More Details

Following [22, 5, 15, 27] we utilize rings defined by cyclotomic polynomials, $\mathbb{A} = \mathbb{Z}[X]/\Phi_m(X)$. We let \mathbb{A}_q denote the set of elements of this ring reduced modulo various (possibly composite) moduli q . The ring \mathbb{A} is the ring of integers of the m th cyclotomic number field K .

A.1 Plaintext Slots

In our scheme plaintexts will be elements of \mathbb{A}_2 , and the polynomial $\Phi_m(X)$ factors modulo 2 into ℓ irreducible factors, $\Phi_m(X) = F_1(X) \cdot F_2(X) \cdots F_\ell(X) \pmod{2}$, all of degree $d = \phi(m)/\ell$. Just as in [5, 15, 27] each factor corresponds to a “plaintext slot”. That is, we view a polynomial $a \in \mathbb{A}_2$ as representing an ℓ -vector $(a \bmod F_i)_{i=1}^\ell$.

It is standard fact that the Galois group $\mathcal{G}al = \mathcal{G}al(\mathbb{Q}(\zeta_m)/\mathbb{Q})$ consists of the mappings $\kappa_k : a(X) \mapsto a(x^k) \bmod \Phi_m(X)$ for all k co-prime with m , and that it is isomorphic to $(\mathbb{Z}/m\mathbb{Z})^*$. As noted in [15], for each $i, j \in \{1, 2, \dots, \ell\}$ there is an element $\kappa_k \in \mathcal{G}al$ which sends an element in slot i to an element in slot j . Namely, if $b = \kappa_i(a)$ then the element in the j ’th slot of b is the same as that in the i ’th slot of a . In addition $\mathcal{G}al$ contains the Frobenius elements, $X \mapsto X^{2^i}$, which also act as Frobenius on the individual slots separately.

For the purpose of implementing AES we will be specifically interested in arithmetic in \mathbb{F}_{2^8} (represented as $\mathbb{F}_{2^8} = \mathbb{F}_2[X]/G(X)$ with $G(X) = X^8 + X^4 + X^3 + X + 1$). We choose the parameters so that d is divisible by 8, so \mathbb{F}_{2^d} includes \mathbb{F}_{2^8} as a subfield. This lets us think of the plaintext space as containing ℓ -vectors over \mathbb{F}_{2^n} .

A.2 Canonical Embedding Norm

Following [22], we use as the “size” of a polynomial $a \in \mathbb{A}$ the l_∞ norm of its canonical embedding. Recall that the canonical embedding of $a \in \mathbb{A}$ into $\mathbb{C}^{\phi(m)}$ is the $\phi(m)$ -vector of complex numbers $\sigma(a) = (a(\zeta_m^i))_i$ where ζ_m is a complex primitive m -th root of unity and the indexes i range over all of $(\mathbb{Z}/m\mathbb{Z})^*$. We call the norm of $\sigma(a)$ the *canonical embedding norm* of a , and denote it by

$$\|a\|_\infty^{\text{can}} = \|\sigma(a)\|_\infty.$$

We will make use of the following properties of $\|\cdot\|_\infty^{\text{can}}$:

- For all $a, b \in \mathbb{A}$ we have $\|a \cdot b\|_\infty^{\text{can}} \leq \|a\|_\infty^{\text{can}} \cdot \|b\|_\infty^{\text{can}}$.
- For all $a \in \mathbb{A}$ we have $\|a\|_\infty^{\text{can}} \leq \|a\|_1$.
- There is a ring constant c_m (depending only on m) such that $\|a\|_\infty \leq c_m \cdot \|a\|_\infty^{\text{can}}$ for all $a \in \mathbb{A}$.

The ring constant c_m is defined by $c_m = \|\text{CRT}_m^{-1}\|_\infty$ where CRT_m is the CRT matrix for m , i.e. the Vandermonde matrix over the complex primitive m -th roots of unity. Asymptotically the value c_m can grow super-polynomially with m , but for the “small” values of m one would use in practice values of c_m can be evaluated directly. See [11] for a discussion of c_m .

Canonical Reduction. When working with elements in \mathbb{A}_q for some integer modulus q , we sometimes need a version of the canonical embedding norm that plays nice with reduction modulo q . Following [15], we define the *canonical embedding norm reduced modulo q* of an element $a \in \mathbb{A}$ as the smallest canonical embedding norm of any a' which is congruent to a modulo q . We denote it as

$$|a|_q^{\text{can}} \stackrel{\text{def}}{=} \min\{ \|a'\|_\infty^{\text{can}} : a' \in \mathbb{A}, a' \equiv a \pmod{q} \}.$$

We sometimes also denote the polynomial where the minimum is obtained by $[a]_q^{\text{can}}$, and call it the *canonical reduction* of a modulo q . Neither the canonical embedding norm nor the canonical reduction is used in the scheme itself, it is only in the analysis of it that we will need them. We note that (trivially) we have $|a|_q^{\text{can}} \leq \|a\|_\infty^{\text{can}}$.

A.3 Double CRT Representation

As noted in Section 2, we usually represent an element $a \in A_q$ via double-CRT representation, with respect to both the polynomial factor of $\Phi_m(X)$ and the integer factors of q . Specifically, we assume that $\mathbb{Z}/q\mathbb{Z}$ contains a primitive m -th root of unity (call it ζ), so $\Phi_m(X)$ factors modulo q to linear terms $\Phi_m(X) = \prod_{i \in (\mathbb{Z}/m\mathbb{Z})^*} (X - \zeta^i) \pmod{q}$. We also denote q 's prime factorization by $q = \prod_{i=0}^t p_i$. Then a polynomial $a \in \mathbb{A}_q$ is represented as the $(t+1) \times \phi(m)$ matrix of its evaluation at the roots of $\Phi_m(X)$ modulo p_i for $i = 0, \dots, t$:

$$\text{dble-CRT}^t(a) = (a(\zeta^j) \pmod{p_i})_{0 \leq i \leq t, j \in (\mathbb{Z}/m\mathbb{Z})^*}.$$

The double CRT representation can be computed using $t+1$ invocations of the FFT algorithm modulo the p_i , picking only the FFT coefficients which correspond to elements in $(\mathbb{Z}/m\mathbb{Z})^*$. To invert this representation we invoke the inverse FFT algorithm $t+1$ times on a vector of length m consisting of the thinned out values padded with zeros, then apply the Chinese Remainder Theorem, and then reduce modulo $\Phi_m(X)$ and q .

Addition and multiplication in \mathbb{A}_q can be computed as component-wise addition and multiplication of the entries in the two tables (modulo the appropriate primes p_i),

$$\begin{aligned} \text{dble-CRT}^t(a + b) &= \text{dble-CRT}^t(a) + \text{dble-CRT}^t(b) \\ \text{dble-CRT}^t(a \cdot b) &= \text{dble-CRT}^t(a) \cdot \text{dble-CRT}^t(b). \end{aligned}$$

Also, for an element of the Galois group $\kappa_k \in \mathcal{G}\mathcal{A}$ (which maps $a(X) \in \mathbb{A}$ to $a(X^k) \pmod{\Phi_m(X)}$), we can evaluate $\kappa_k(a)$ on the double-CRT representation of a just by permuting the columns in the matrix, sending each column j to column $j \cdot k \pmod{m}$.

A.4 Sampling From \mathbb{A}_q

At various points we will need to sample from \mathbb{A}_q with different distributions, as described below. We denote choosing the element $a \in \mathbb{A}$ according to distribution \mathcal{D} by $a \leftarrow \mathcal{D}$. The distributions below are described as over $\phi(m)$ -vectors, but we always consider them as distributions over the ring \mathbb{A} , by identifying a polynomial $a \in \mathbb{A}$ with its coefficient vector.

The uniform distribution \mathcal{U}_q : This is just the uniform distribution over $(\mathbb{Z}/q\mathbb{Z})^{\phi(m)}$, which we identify with $(\mathbb{Z} \cap (-q/2, q/2])^{\phi(m)}$. Note that it is easy to sample from \mathcal{U}_q directly in double-CRT representation.

The “discrete Gaussian” $\mathcal{DG}_q(\sigma^2)$: Let $\mathcal{N}(0, \sigma^2)$ denote the normal (Gaussian) distribution on real numbers with zero-mean and variance σ^2 , we use drawing from $\mathcal{N}(0, \sigma^2)$ and rounding to the nearest integer as an approximation to the discrete Gaussian distribution. Namely, the distribution $\mathcal{DG}_q(\sigma^2)$ draws a real ϕ -vector according to $\mathcal{N}(0, \sigma^2)^{\phi(m)}$, rounds it to the nearest integer vector, and outputs that integer vector reduced modulo q (into the interval $(-q/2, q/2]$).

Sampling small polynomials, $\mathcal{ZO}(p)$ and $\mathcal{HWT}(h)$: These distributions produce vectors in $\{0, \pm 1\}^{\phi(m)}$.

For a real parameter $\rho \in [0, 1]$, $\mathcal{ZO}(p)$ draws each entry in the vector from $\{0, \pm 1\}$, with probability $\rho/2$ for each of -1 and $+1$, and probability of being zero $1 - \rho$.

For an integer parameter $h \leq \phi(m)$, the distribution $\mathcal{HWT}(h)$ chooses a vector uniformly at random from $\{0, \pm 1\}^{\phi(m)}$, subject to the conditions that it has exactly h nonzero entries.

A.5 Canonical embedding norm of random polynomials

In the coming sections we will need to bound the canonical embedding norm of polynomials that are produced by the distributions above, as well as products of such polynomials. In some cases it is possible to analyze the norm rigorously using Chernoff and Hoeffding bounds, but to set the parameters of our scheme we instead use a heuristic approach that yields better constants:

Let $a \in \mathbb{A}$ be a polynomial that was chosen by one of the distributions above, hence all the (nonzero) coefficients in a are IID (independently identically distributed). For a complex primitive m -th root of unity ζ_m , the evaluation $a(\zeta_m)$ is the inner product between the coefficient vector of a and the fixed vector $\mathbf{z}_m = (1, \zeta_m, \zeta_m^2, \dots)$, which has Euclidean norm exactly $\sqrt{\phi(m)}$. Hence the random variable $a(\zeta_m)$ has variance $V = \sigma^2 \phi(m)$, where σ^2 is the variance of each coefficient of a . Specifically, when $a \leftarrow \mathcal{U}_q$ then each coefficient has variance $q^2/12$, so we get variance $V_U = q^2 \phi(m)/12$. When $a \leftarrow \mathcal{DG}_q(\sigma^2)$ we get variance $V_G \approx \sigma^2 \phi(m)$, and when $a \leftarrow \mathcal{ZO}(\rho)$ we get variance $V_Z = \rho \phi(m)$. When choosing $a \leftarrow \mathcal{HWT}(h)$ we get a variance of $V_H = h$ (but not $\phi(m)$, since a has only h nonzero coefficients).

Moreover, the random variable $a(\zeta_m)$ is a sum of many IID random variables, hence by the law of large numbers it is distributed similarly to a complex Gaussian random variable of the specified variance.⁵ We therefore use $6\sqrt{V}$ (i.e. six standard deviations) as a high-probability bound on the size of $a(\zeta_m)$. Since the evaluation of a at all the roots of unity obeys the same bound, we use six standard deviations as our bound on the canonical embedding norm of a . (We chose six standard deviations since $\text{erfc}(6) \approx 2^{-55}$, which is good enough for us even when using the union bound and multiplying it by $\phi(m) \approx 2^{16}$.)

In many cases we need to bound the canonical embedding norm of a product of two such “random polynomials”. In this case our task is to bound the magnitude of the product of two random variables, both are distributed close to Gaussians, with variances σ_a^2, σ_b^2 , respectively. For this case we use $16\sigma_a\sigma_b$ as our

⁵The mean of $a(\zeta_m)$ is zero, since the coefficients of a are chosen from a zero-mean distribution.

bound, since $\text{erfc}(4) \approx 2^{-25}$, so the probability that both variables exceed their standard deviation by more than a factor of four is roughly 2^{-50} .

B The Basic Scheme

We now define our leveled HE scheme on L levels; including the Modulus-Switching and Key-Switching operations and the procedures for KeyGen, Enc, Dec, and for Add, Mult, Scalar-Mult, and Automorphism.

Recall that a ciphertext vector \mathbf{c} in the cryptosystem is a valid encryption of $a \in \mathbb{A}$ with respect to secret key \mathbf{s} and modulus q if $[[\langle \mathbf{c}, \mathbf{s} \rangle]_q]_2 = a$, where the inner product is over $\mathbb{A} = \mathbb{Z}[X]/\Phi_m(X)$, the operation $[\cdot]_q$ denotes modular reduction in coefficient representation into the interval $(-q/2, +q/2]$, and we require that the “noise” $[\langle \mathbf{c}, \mathbf{s} \rangle]_q$ is sufficiently small (in canonical embedding norm reduced mod q). In our implementation a “normal” ciphertext is a 2-vector $\mathbf{c} = (c_0, c_1)$, and a “normal” secret key is of the form $\mathbf{s} = (1, -\mathbf{s})$, hence decryption takes the form

$$a \leftarrow [c_0 - c_1 \cdot \mathbf{s}]_q \bmod 2. \quad (2)$$

B.1 Our Moduli Chain

We define the chain of moduli for our depth- L homomorphic evaluation by choosing L “small primes” p_0, p_1, \dots, p_{L-1} and the t ’th modulus in our chain is defined as $q_t = \prod_{j=0}^t p_j$. (The sizes will be determined later.) The primes p_i ’s are chosen so that for all i , $\mathbb{Z}/p_i\mathbb{Z}$ contains a primitive m -th root of unity. Hence we can use our double-CRT representation for all \mathbb{A}_{q_t} .

This choice of moduli makes it easy to get a level- $(t-1)$ representation of $a \in \mathbb{A}$ from its level- t representation. Specifically, given the level- t double-CRT representation $\text{dble-CRT}^t(a)$ for some $a \in \mathbb{A}_{q_t}$, we can simply remove from the matrix the row corresponding to the last small prime p_t , thus obtaining a level- $(t-1)$ representation of $a \bmod q_{t-1} \in \mathbb{A}_{q_{t-1}}$. Similarly we can get the double-CRT representation for lower levels by removing more rows. By a slight abuse of notation we write $\text{dble-CRT}^{t'}(a) = \text{dble-CRT}^t(a) \bmod q_{t'}$ for $t' < t$.

Recall that encryption produces ciphertext vectors valid with respect to the largest modulus q_{L-1} in our chain, and we obtain ciphertext vectors valid with respect to smaller moduli whenever we apply modulus-switching to decrease the noise magnitude. As described in Section 3.3, our implementation *dynamically* adjust levels, performing modulus switching when the dynamically-computed noise estimate becomes too large. Hence each ciphertext in our scheme is tagged with both its level t (pinpointing the modulus q_t relative to which this ciphertext is valid), and an estimate ν on the noise magnitude in this ciphertext. In other words, a ciphertext is a triple (\mathbf{c}, t, ν) with $0 \leq t \leq L-1$, \mathbf{c} a vector over \mathbb{A}_{q_t} , and ν a real number which is used as our noise estimate.

B.2 Modulus Switching

The operation $\text{SwitchModulus}(\mathbf{c})$ takes the ciphertext $\mathbf{c} = ((c_0, c_1), t, \nu)$ defined modulo q_t and produces a ciphertext $\mathbf{c}' = ((c'_0, c'_1), t-1, \nu')$ defined modulo q_{t-1} . Such that $[c_0 - \mathbf{s} \cdot c_1]_{q_t} \equiv [c'_0 - \mathbf{s} \cdot c'_1]_{q_{t-1}} \pmod{2}$, and ν' is smaller than ν . This procedure makes use of the function $\text{Scale}(x, q, q')$ that takes an element $x \in \mathbb{A}_q$ and returns an element $y \in \mathbb{A}_{q'}$ such that in coefficient representation it holds that $y \equiv x \pmod{2}$, and y is the closest element to $(q'/q) \cdot x$ that satisfies this mod-2 condition.

To maintain the noise estimate, the procedure uses the pre-set ring-constant c_m (cf. Appendix A.2) and also a pre-set constant B_{scale} which is meant to bound the magnitude of the added noise term from this operation. It works as follows:

SwitchModulus($(c_0, c_1), t, \nu$):

1. If $t < 1$ then abort; // Sanity check
2. $\nu' \leftarrow \frac{q_{t-1}}{q_t} \cdot \nu + B_{\text{scale}}$; // Scale down the noise estimate
3. If $\nu' > q_{t-1}/2c_m$ then abort; // Another sanity check
4. $c'_i \leftarrow \text{Scale}(c_i, q_t, q_{t-1})$ for $i = 0, 1$; // Scale down the vector
5. Output $((c'_0, c'_1), t - 1, \nu')$.

The constant B_{scale} is set as $B_{\text{scale}} = 2\sqrt{\phi(m)/3} \cdot (8\sqrt{h} + 3)$, where h is the Hamming weight of the secret key. (In our implementation we use $h = 64$, so we get $B_{\text{scale}} \approx 77\sqrt{\phi(m)}$.) To justify this choice, we apply to the proof of the modulus switching lemma from [15, Lemma 13] (in the full version), relative to the canonical embedding norm. In that proof it is shown that when the noise magnitude in the input ciphertext $\mathbf{c} = (c_0, c_1)$ is bounded by ν , then the noise magnitude in the output vector $\mathbf{c}' = (c'_0, c'_1)$ is bounded by $\nu' = \frac{q_{t-1}}{q_t} \cdot \nu + \|\langle \mathbf{s}, \tau \rangle\|_{\infty}^{\text{can}}$, provided that the last quantity is smaller than $q_{t-1}/2$.

Above τ is the “rounding error” vector, namely $\tau \stackrel{\text{def}}{=} (\tau_0, \tau_1) = (c'_0, c'_1) - \frac{q_{t-1}}{q_t}(c_0, c_1)$. Heuristically assuming that τ behaves as if its coefficients are chosen uniformly in $[-1, +1]$, the evaluation $\tau_i(\zeta)$ at an m -th root of unity ζ_m is distributed close to a Gaussian complex with variance $\phi(m)/3$. Also, \mathbf{s} was drawn from $\mathcal{HWT}(h)$ so $\mathbf{s}(\zeta_m)$ is distributed close to a Gaussian complex with variance h . Hence we expect $\tau_1(\zeta)\mathbf{s}(\zeta)$ to have magnitude at most $16\sqrt{\phi(m)/3 \cdot h}$ (recall that we use $h = 64$). We can similarly bound $\tau_0(\zeta_m)$ by $6\sqrt{\phi(m)/3}$, and therefore the evaluation of $\langle \mathbf{s}, \tau \rangle$ at ζ_m is bounded in magnitude (whp) by:

$$16\sqrt{\phi(m)/3 \cdot h} + 6\sqrt{\phi(m)/3} = 2\sqrt{\phi(m)/3} \cdot (8\sqrt{h} + 3) \approx 77\sqrt{\phi(m)} = B_{\text{scale}} \quad (3)$$

B.3 Key Switching

After some homomorphic evaluation operations we have on our hands not a “normal” ciphertext which is valid relative to “normal” secret key, but rather an “extended ciphertext” $((d_0, d_1, d_2), q_t, \nu)$ which is valid with respect to an “extended secret key” $\mathbf{s}' = (1, -\mathbf{s}, -\mathbf{s}')$. Namely, this ciphertext encrypts the plaintext $a \in \mathbb{A}$ via

$$a = \left[[d_0 - \mathbf{s} \cdot d_1 - \mathbf{s}' \cdot d_2]_{q_t} \right]_2$$

and the magnitude of the noise $[d_0 - \mathbf{s} \cdot d_1 - d_2 \cdot \mathbf{s}']_{q_t}$ is bounded by ν . In our implementation, the component \mathbf{s} is always the same element $\mathbf{s} \in \mathbb{A}$ that was drawn from $\mathcal{HWT}(h)$ during key generation, but \mathbf{s}' can vary depending on the operation. (See the description of multiplication and automorphisms below.)

To enable that translation, we use some “key switching matrices” that are included in the public key. (In our implementation these “matrices” have dimension 2×1 , i.e., they consist of only two elements from \mathbb{A} .) As explained in Section 3.1, we save on space and time by artificially “boosting” the modulus we use from q_t up to $P \cdot q_t$ for some “large” modulus P . We note that in order to represent elements in \mathbb{A}_{Pq_t} using our dble-CRT representation we need to choose P so that $\mathbb{Z}/P\mathbb{Z}$ also has primitive m -th roots of unity. (In fact in our implementation we pick P to be a prime.)

The key-switching “matrix”. Denote by $Q = P \cdot q_{L-2}$ the largest modulus relative to which we need to generate key-switching matrices. To generate the key-switching matrix from $\mathbf{s}' = (1, -\mathfrak{s}, -\mathfrak{s}')$ to $\mathbf{s} = (1, -\mathfrak{s})$ (note that both keys share the same element \mathfrak{s}), we choose two element, one uniform and the other from our “discrete Gaussian”,

$$a_{\mathfrak{s}, \mathfrak{s}'} \leftarrow \mathcal{U}_Q \text{ and } e_{\mathfrak{s}, \mathfrak{s}'} \leftarrow \mathcal{DG}_Q(\sigma^2),$$

where the variance σ is a global parameter (that we later set as $\sigma = 3.2$). The “key switching matrix” then consists of the single column vector

$$W[\mathbf{s}' \rightarrow \mathbf{s}] = \begin{pmatrix} b_{\mathfrak{s}, \mathfrak{s}'} \\ a_{\mathfrak{s}, \mathfrak{s}'} \end{pmatrix}, \text{ where } b_{\mathfrak{s}, \mathfrak{s}'} \stackrel{\text{def}}{=} [\mathfrak{s} \cdot a_{\mathfrak{s}, \mathfrak{s}'} + 2e_{\mathfrak{s}, \mathfrak{s}'} + P\mathfrak{s}']_Q. \quad (4)$$

Note that W above is defined modulo $Q = Pq_{L-2}$, but we need to use it relative to $Q_t = Pq_t$ for whatever the current level t is. Hence before applying the key switching procedure at level t , we reduce W modulo Q_t to get $W_t \stackrel{\text{def}}{=} [W]_{Q_t}$. It is important to note that since Q_t divides Q then W_t is indeed a key-switching matrix. Namely it is of the form $(b, a)^T$ with $a \in \mathcal{U}_{Q_t}$ and $b = [\mathfrak{s} \cdot a + 2e_{\mathfrak{s}, \mathfrak{s}'} + P\mathfrak{s}']_{Q_t}$ (with respect to the same element $e_{\mathfrak{s}, \mathfrak{s}'} \in \mathbb{A}$ from above).

The SwitchKey procedure. Given the extended ciphertext $\mathbf{c} = ((d_0, d_1, d_2), t, \nu)$ and the key-switching matrix $W_t = (b, a)^T$, the procedure $\text{SwitchKey}_{W_t}(\mathbf{c})$ proceeds as follows:⁶

SwitchKey_(b,a)((d₀, d₁, d₂), t, ν):

1. Set $\begin{pmatrix} c'_0 \\ c'_1 \end{pmatrix} \leftarrow \left[\begin{pmatrix} Pd_0 & b \\ Pd_1 & a \end{pmatrix} \begin{pmatrix} 1 \\ d_2 \end{pmatrix} \right]_{Q_t}$; // The actual key-switching operation
2. $c''_i \leftarrow \text{Scale}(c'_i, Q_t, q_t)$ for $i = 0, 1$; // Scale the vector back down to q_t
3. $\nu' \leftarrow \nu + B_{\mathbf{K}\mathfrak{s}} \cdot q_t / P + B_{\text{scale}}$; // The constant $B_{\mathbf{K}\mathfrak{s}}$ is determined below
4. Output $((c''_0, c''_1), t, \nu')$.

To argue correctness, observe that although the “actual key switching operation” from above looks superficially different from the standard key-switching operation $\mathbf{c}' \leftarrow W \cdot \mathbf{c}$, it is merely an optimization that takes advantage of the fact that both vectors \mathbf{s}' and \mathbf{s} share the element \mathfrak{s} . Indeed, we have the equality over \mathbb{A}_{Q_t} :

$$\begin{aligned} c'_0 - \mathfrak{s} \cdot c'_1 &= [(P \cdot d_0) + d_2 \cdot b_{\mathfrak{s}, \mathfrak{s}'} - \mathfrak{s} \cdot ((P \cdot d_1) + d_2 \cdot a_{\mathfrak{s}, \mathfrak{s}'})] \\ &= P \cdot (d_0 - \mathfrak{s} \cdot d_1 - \mathfrak{s}' d_2) + 2 \cdot d_2 \cdot e_{\mathfrak{s}, \mathfrak{s}'}, \end{aligned}$$

so as long as both sides are smaller than Q_t we have the same equality also over \mathbb{A} (without the mod- Q_t reduction), which means that we get

$$[c'_0 - \mathfrak{s} \cdot c'_1]_{Q_t} = [P \cdot (d_0 - \mathfrak{s} \cdot d_1 - \mathfrak{s}' d_2) + 2 \cdot d_2 \cdot e_{\mathfrak{s}, \mathfrak{s}'}]_{Q_t} \equiv [d_0 - \mathfrak{s} \cdot d_1 - \mathfrak{s}' d_2]_{Q_t} \pmod{2}.$$

To analyze the size of the added term $2d_2 e_{\mathfrak{s}, \mathfrak{s}'}$, we can assume heuristically that d_2 behaves like a uniform polynomial drawn from \mathcal{U}_{q_t} , hence $d_2(\zeta_m)$ for a complex root of unity ζ_m is distributed close to a complex Gaussian with variance $q_t^2 \phi(m)/12$. Similarly $e_{\mathfrak{s}, \mathfrak{s}'}(\zeta_m)$ is distributed close to a complex Gaussian with

⁶For simplicity we describe the SwitchKey procedure as if it always switches back to mod- q_t , but in reality if the noise estimate is large enough then it can switch directly to q_{t-1} instead.

variance $\sigma^2\phi(m)$, so $2d_2(\zeta)\epsilon(\zeta)$ can be modeled as a product of two Gaussians, and we expect that with overwhelming probability it remains smaller than $2 \cdot 16 \cdot \sqrt{q_t^2\phi(m)/12 \cdot \sigma^2\phi(m)} = \frac{16}{\sqrt{3}} \cdot \sigma q_t\phi(m)$. This yields a heuristic bound $16/\sqrt{3} \cdot \sigma\phi(m) \cdot q_t = B_{K_S} \cdot q_t$ on the canonical embedding norm of the added noise term, and if the total noise magnitude does not exceed $Q_t/2c_m$ then also in coefficient representation everything remains below $Q_t/2$. Thus our constant B_{K_S} is set as

$$\frac{16\sigma\phi(m)}{\sqrt{3}} \approx 9\sigma\phi(m) = B_{K_S} \quad (5)$$

Finally, dividing by P (which is the effect of the Scale operation), we obtain the final ciphertext that we require, and the noise magnitude is divided by P (except for the added B_{scale} term).

B.4 Key-Generation, Encryption, and Decryption

The procedures below depend on many parameters, h, σ, m , the primes p_i and P , etc. These parameters will be determined later.

KeyGen(): Given the parameters, the key generation procedure chooses a low-weight secret key and then generates an LWE instance relative to that secret key. Namely, we choose

$$\mathfrak{s} \leftarrow \mathcal{HWT}(h), \quad a \leftarrow \mathcal{U}_{q_{L-1}}, \quad \text{and } e \leftarrow \mathcal{DG}_{q_{L-1}}(\sigma^2)$$

Then sets the secret key as \mathfrak{s} and the public key as (a, b) where $b = [a \cdot s + 2e]_{q_{L-1}}$.

In addition, the key generation procedure adds to the public key some key-switching “matrices”, as described in Appendix B.3. Specifically the matrix $W[\mathfrak{s}^2 \rightarrow \mathfrak{s}]$ for use in multiplication, and some matrices $W[\kappa_i(\mathfrak{s}) \rightarrow \mathfrak{s}]$ for use in automorphisms, for $\kappa_i \in \mathcal{Gal}$ whose indexes generates $(\mathbb{Z}/m\mathbb{Z})^*$ (including in particular κ_2).

Enc_{pt}(\mathbf{m}): To encrypt an element $m \in \mathbb{A}_2$, we choose one “small polynomial” (with $0, \pm 1$ coefficients) and two Gaussian polynomials (with variance σ^2),

$$v \leftarrow \mathcal{ZO}(0.5) \quad \text{and } e_0, e_1 \leftarrow \mathcal{DG}_{q_{L-1}}(\sigma^2)$$

Then we set $c_0 = b \cdot v + 2 \cdot e_0 + m$, $c_1 = a \cdot v + 2 \cdot e_1$, and set the initial ciphertext as $\mathfrak{c}' = (c_0, c_1, L-1, B_{\text{clean}})$, where B_{clean} is a parameter that we determine below.

The noise magnitude in this ciphertext (B_{clean}) is a little larger than what we would like, so before we start computing on it we do one modulus-switch. That is, the encryption procedure sets $\mathfrak{c} \leftarrow \text{SwitchModulus}(\mathfrak{c}')$ and outputs \mathfrak{c} . We can deduce a value for B_{clean} as follows:

$$\begin{aligned} |c_0 - \mathfrak{s} \cdot c_1|_{q_t}^{\text{can}} &\leq \|c_0 - \mathfrak{s} \cdot c_1\|_{\infty}^{\text{can}} \\ &= \|((a \cdot s + 2 \cdot e) \cdot v + 2 \cdot e_0 + \mathbf{m} - (a \cdot v + 2 \cdot e_1) \cdot \mathfrak{s})\|_{\infty}^{\text{can}} \\ &= \|\mathbf{m} + 2 \cdot (e \cdot v + e_0 - e_1 \cdot \mathfrak{s})\|_{\infty}^{\text{can}} \\ &\leq \|\mathbf{m}\|_{\infty}^{\text{can}} + 2 \cdot (\|e \cdot v\|_{\infty}^{\text{can}} + \|e_0\|_{\infty}^{\text{can}} + \|e_1 \cdot \mathfrak{s}\|_{\infty}^{\text{can}}) \end{aligned}$$

Using our complex Gaussian heuristic from Appendix A.5, we can bound the canonical embedding norm of the randomized terms above by

$$\|e \cdot v\|_{\infty}^{\text{can}} \leq 16\sigma\phi(m)/\sqrt{2}, \quad \|e_0\|_{\infty}^{\text{can}} \leq 6\sigma\sqrt{\phi(m)}, \quad \|e_1 \cdot \mathfrak{s}\|_{\infty}^{\text{can}} \leq 16\sigma\sqrt{h \cdot \phi(m)}$$

Also, the norm of the input message m is clearly bounded by $\phi(m)$, hence (when we substitute our parameters $h = 64$ and $\sigma = 3.2$) we get the bound

$$\phi(m) + 32\sigma\phi(m)/\sqrt{2} + 12\sigma\sqrt{\phi(m)} + 32\sigma\sqrt{h \cdot \phi(m)} \approx 74\phi(m) + 858\sqrt{\phi(m)} = B_{\text{clean}} \quad (6)$$

Our goal in the initial modulus switching from q_{L-1} to q_{L-2} is to reduce the noise from its initial level of $B_{\text{clean}} = \Theta(\phi(m))$ to our base-line bound of $B = \Theta(\sqrt{\phi(m)})$ which is determined in Equation (12) below.

Dec_{pt}(c): Decryption of a ciphertext (c_0, c_1, t, ν) at level t is performed by setting $m' \leftarrow [c_0 - \mathfrak{s} \cdot c_1]_{q_t}$, then converting m' to coefficient representation and outputting $m' \bmod 2$. This procedure works when $c_m \cdot \nu < q_t/2$, so this procedure only applies when the constant c_m for the field \mathbb{A} is known and relatively small (which as we mentioned above will be true for all practical parameters). Also, we must pick the smallest prime $q_0 = p_0$ large enough, as described in Appendix C.2.

B.5 Homomorphic Operations

Add(c, c'): Given two ciphertexts $\mathbf{c} = ((c_0, c_1), t, \nu)$ and $\mathbf{c}' = ((c'_0, c'_1), t', \nu')$, representing messages $\mathbf{m}, \mathbf{m}' \in \mathbb{A}_2$, this algorithm forms a ciphertext $\mathbf{c}_a = ((a_0, a_1), t_a, \nu_a)$ which encrypts the message $\mathbf{m}_a = \mathbf{m} + \mathbf{m}'$.

If the two ciphertexts do not belong to the same level then we reduce the larger one modulo the smaller of the two moduli, thus bringing them to the same level. (This simple modular reduction works as long as the noise magnitude is smaller than the smaller of the two moduli, if this condition does not hold then we need to do modulus switching rather than simple modular reduction.) Once the two ciphertexts are at the same level (call it t''), we just add the two ciphertext vectors and two noise estimates to get

$$\mathbf{c}_a = \left(([c_0 + c'_0]_{q_{t''}}, [c_1 + c'_1]_{q_{t''}}), t'', \nu + \nu' \right).$$

Mult(c, c'): Given two ciphertexts representing messages $\mathbf{m}, \mathbf{m}' \in \mathbb{A}_2$, this algorithm forms a ciphertext encrypts the message $\mathbf{m} \cdot \mathbf{m}'$.

We begin by ensuring that the noise magnitude in both ciphertexts is smaller than the pre-set constant B (which is our base-line bound and is determined in Equation (12) below), performing modulus-switching as needed to ensure this condition. Then we bring both ciphertexts to the same level by reducing modulo the smaller of the two moduli (if needed). Once both ciphertexts have small noise magnitude and the same level we form the extended ciphertext (essentially performing the tensor product of the two) and apply key-switching to get back a normal ciphertext. A pseudo-code description of this procedure is given below.

Mult(c, c'):

1. While $\nu(\mathbf{c}) > B$ do $\mathbf{c} \leftarrow \text{SwitchModulus}(\mathbf{c})$; // $\nu(\mathbf{c})$ is the noise estimate in \mathbf{c}
2. While $\nu(\mathbf{c}') > B$ do $\mathbf{c}' \leftarrow \text{SwitchModulus}(\mathbf{c}')$; // $\nu(\mathbf{c}')$ is the noise estimate in \mathbf{c}'
3. Bring \mathbf{c}, \mathbf{c}' to the same level t by reducing modulo the smaller of the two moduli
Denote after modular reduction $\mathbf{c} = ((c_0, c_1), t, \nu)$ and $\mathbf{c}' = ((c'_0, c'_1), t, \nu')$
4. Set $(d_0, d_1, d_2) \leftarrow (c_0 \cdot c'_0, c_1 \cdot c'_0 + c_0 \cdot c'_1, -c_1 \cdot c'_1)$;
Denote $\mathbf{c}'' = ((d_0, d_1, d_2), t, \nu \cdot \nu')$
5. Output $\text{SwitchKey}_{W[\mathfrak{s}^2 \rightarrow \mathfrak{s}]}(\mathbf{c}'')$ // Convert to “normal” ciphertext

We stress that *the only place* where we force modulus switching is before the multiplication operation. In all other operations we allow the noise to grow, and it will be reduced back the first time it is input to a multiplication operation. We also note that we may need to apply modulus switching more than once before the noise is small enough.

Scalar-Mult(\mathbf{c}, α): Given a ciphertext $\mathbf{c} = (c_0, c_1, t, \nu)$ representing the message \mathbf{m} , and an element $\alpha \in \mathbb{A}_2$ (represented as a polynomial modulo 2 with coefficients in $\{-1, 0, 1\}$), this algorithm forms a ciphertext $\mathbf{c}_m = (a_0, a_1, t_m, \nu_m)$ which encrypts the message $\mathbf{m}_m = \alpha \cdot \mathbf{m}$. This procedure is needed in our implementation of homomorphic AES, and is of more general interest in general computation over finite fields.

The algorithm makes use of a procedure $\text{Randomize}(\alpha)$ which takes α and replaces each non-zero coefficients with a coefficients chosen at random from $\{-1, 1\}$. To multiply by α , we set $\beta \leftarrow \text{Randomize}(\alpha)$ and then just multiply both c_0 and c_1 by β . Using the same argument as we used in Appendix A.5 for the distribution $\mathcal{HWT}(h)$, here too we can bound the norm of β by $\|\beta\|_\infty^{\text{can}} \leq 6\sqrt{\text{Wt}(\alpha)}$ where $\text{Wt}(\alpha)$ is the number of nonzero coefficients of α . Hence we multiply the noise estimate by $6\sqrt{\text{Wt}(\alpha)}$, and output the resulting ciphertext $\mathbf{c}_m = (c_0 \cdot \beta, c_1 \cdot \beta, t, \nu \cdot 6\sqrt{\text{Wt}(\alpha)})$.

Automorphism(\mathbf{c}, κ): In the main body we explained how permutations on the plaintext slots can be realized via using elements $\kappa \in \mathcal{Gal}$; we also require the application of such automorphism to implement the Frobenius maps in our AES implementation.

For each κ that we want to use, we need to include in the public key the “matrix” $W[\kappa(\mathfrak{s}) \rightarrow \mathfrak{s}]$. Then, given a ciphertext $\mathbf{c} = (c_0, c_1, t, \nu)$ representing the message \mathbf{m} , the function $\text{Automorphism}(\mathbf{c}, \kappa)$ produces a ciphertext $\mathbf{c}' = (c'_0, c'_1, t, \nu')$ which represents the message $\kappa(\mathbf{m})$. We first set an “extended ciphertext” by setting

$$d_0 = \kappa(c_0), \quad d_1 \leftarrow 0, \quad \text{and } d_2 \leftarrow \kappa(c_1)$$

and then apply key switching to the extended ciphertext $((d_0, d_1, d_2), t, \nu)$ using the “matrix” $W[\kappa(\mathfrak{s}) \rightarrow \mathfrak{s}]$.

C Security Analysis and Parameter Settings

Below we derive the concrete parameters for use in our implementation. We begin in Appendix C.1 by deriving a lower-bound on the dimension N of the LWE problem underlying our key-switching matrices, as a function of the modulus and the noise variance. (This will serve as a lower-bound on $\phi(m)$ for our choice of the ring polynomial $\Phi_m(X)$.) Then in Appendix C.2 we derive a lower bound on the size of the largest modulus Q in our implementation, in terms of the noise variance and the dimension N . Then in Appendix C.3 we choose a value for the noise variance (as small as possible subject to some nominal security concerns), solve the somewhat circular constraints on N and Q , and set all the other parameters.

C.1 Lower-Bounding the Dimension

Below we apply to the LWE-security analysis of Lindner and Peikert [20], together with a few (arguably justifiable) assumptions, to analyze the dimension needed for different security levels. The analysis below assumes that we are given the modulus Q and noise variance σ^2 for the LWE problem (i.e., the noise is chosen from a discrete Gaussian distribution modulo Q with variance σ^2 in each coordinate). The goal is to derive a lower-bound on the dimension N required to get any given security level. The first assumption that we make, of course, is that the Lindner-Peikert analysis — which was done in the context of standard LWE — applies also for our ring-LWE case. We also make the following extra assumptions:

- We assume that (once σ is not too tiny), the security depends on the ratio Q/σ and not on Q and σ separately. Nearly all the attacks and hardness results in the literature support this assumption, with the exception of the Arora-Ge attack [2] (that works whenever σ is very small, regardless of Q).
- The analysis in [20] devised an experimental formula for the time that it takes to get a particular quality of reduced basis (i.e., the parameter δ of Gama and Nguyen [12]), then provided another formula for the advantage that the attack can derive from a reduced basis at a given quality, and finally used a computer program to solve these formulas for some given values of N and δ . This provides some time/advantage tradeoff, since obtaining a smaller value of δ (i.e., higher-quality basis) takes longer time and provides better advantage for the attacker.

For our purposes we made the assumption that the best runtime/advantage ratio is achieved in the high-advantage regime. Namely we should spend basically all the attack running time doing lattice reduction, in order to get a good enough basis that will break security with advantage (say) $1/2$. This assumption is consistent with the results that are reported in [20].

- Finally, we assume that to get advantage of close to $1/2$ for an LWE instance with modulus Q and noise σ , we need to be able to reduce the basis well enough until the shortest vector is of size roughly Q/σ . Again, this is consistent with the results that are reported in [20].

Given these assumptions and the formulas from [20], we can now solve the dimension/security tradeoff analytically. Because of the first assumption we might as well simplify the equations and derive our lower bound on N for the case $\sigma = 1$, where the ratio Q/σ is equal to Q . (In reality we will use $\sigma \approx 4$ and increase the modulus by the same 2 bits).

Following Gama-Nguyen [12], recall that a reduced basis $B = (b_1|b_2|\dots|b_m)$ for a dimension- M , determinant- D lattice (with $\|b_1\| \leq \|b_2\| \leq \dots \leq \|b_M\|$), has quality parameter δ if the shortest vector in that basis has norm $\|b_1\| = \delta^M \cdot D^{1/M}$. In other words, the quality of B is defined as $\delta = \|b_1\|^{1/M} / D^{1/M^2}$. The time (in seconds) that it takes to compute a reduced basis of quality δ for a random LWE instance was estimated in [20] to be at least

$$\log(\text{time}) \geq 1.8/\log(\delta) - 110. \quad (7)$$

For a random Q -ary lattice of rank N , the determinant is exactly Q^N whp, and therefore a quality- δ basis has $\|b_1\| = \delta^M \cdot Q^{N/M}$. By our second assumption, we should reduce the basis enough so that $\|b_1\| = Q$, so we need $Q = \delta^M \cdot Q^{N/M}$. The LWE attacker gets to choose the dimension M , and the best choice for this attack is obtained when the right-hand-side of the last equality is minimized, namely for $M = \sqrt{N \log Q / \log \delta}$. This yields the condition

$$\log Q = \log(\delta^M Q^{N/M}) = M \log \delta + (N/M) \log Q = 2\sqrt{N \log Q \log \delta},$$

which we can solve for N to get $N = \log Q / 4 \log \delta$. Finally, we can use Equation (7) to express $\log \delta$ as a function of $\log(\text{time})$, thus getting $N = \log Q \cdot (\log(\text{time}) + 110) / 7.2$. Recalling that in our case we used $\sigma = 1$ (so $Q/\sigma = Q$), we get our lower-bound on N in terms of Q/σ . Namely, to ensure a time/advantage ratio of at least 2^k , we need to set the rank N to be at least

$$N \geq \frac{\log(Q/\sigma)(k + 110)}{7.2} \quad (8)$$

For example, the above formula says that to get 80-bit security level we need to set $N \geq \log(Q/\sigma) \cdot 26.4$, for 100-bit security level we need $N \geq \log(Q/\sigma) \cdot 29.1$, and for 128-bit security level we need $N \geq \log(Q/\sigma) \cdot 33.1$. We comment that these values are indeed consistent with the values reported in [20].

C.1.1 LWE with Sparse Key

The analysis above applies to “generic” LWE instance, but in our case we use very sparse secret keys (with only $h = 64$ nonzero coefficients, all chosen as ± 1). This brings up the question of whether one can get better attacks against LWE instances with a very sparse secret (much smaller than even the noise). We note that Goldwasser et al. proved in [16] that LWE with low-entropy secret is as hard as standard LWE with weaker parameters (for large enough moduli). Although the specific parameters from that proof do not apply to our choice of parameter, it does indicate that weak-secret LWE is not “fundamentally weaker” than standard LWE. In terms of attacks, the only attack that we could find that takes advantage of this sparse key is by applying the reduction technique of Applebaum et al. [1] to switch the key with part of the error vector, thus getting a smaller LWE error.

In a sparse-secret LWE we are given a random N -by- M matrix A (modulo Q), and also an M -vector $\mathbf{y} = [\mathbf{s}A + \mathbf{e}]_Q$. Here the N -vector \mathbf{s} is our very sparse secret, and \mathbf{e} is the error M -vector (which is also short, but not sparse and not as short as \mathbf{s}).

Below let A_1 denotes the first N columns of A , A_2 the next N columns, then A_3, A_4 , etc. Similarly $\mathbf{e}_1, \mathbf{e}_2, \dots$ are the corresponding parts of the error vector and $\mathbf{y}_1, \mathbf{y}_2, \dots$ the corresponding parts of \mathbf{y} . Assuming that A_1 is invertible (which happens with high probability), we can transform this into an LWE instance with respect to secret \mathbf{e}_1 , as follows:

We have $\mathbf{y}_1 = \mathbf{s}A_1 + \mathbf{e}_1$, or alternatively $A_1^{-1}\mathbf{y}_1 = \mathbf{s} + A_1^{-1}\mathbf{e}_1$. Also, for $i > 1$ we have $\mathbf{y}_i = \mathbf{s}A_i + \mathbf{e}_i$, which together with the above gives $A_iA_1^{-1}\mathbf{y}_1 - \mathbf{y}_i = A_iA_1^{-1}\mathbf{e}_1 - \mathbf{e}_i$. Hence if we denote

$$B_1 \stackrel{\text{def}}{=} A_1^{-1}, \quad \text{and for } i > 1 \quad B_i \stackrel{\text{def}}{=} A_iA_1^{-1},$$

$$\text{and similarly } \mathbf{z}_1 = A_1^{-1}\mathbf{y}_1, \quad \text{and for } i > 1 \quad \mathbf{z}_i \stackrel{\text{def}}{=} A_iA_1^{-1}\mathbf{y}_i,$$

and then set $B \stackrel{\text{def}}{=} (B_1^t | B_2^t | B_3^t | \dots)$ and $\mathbf{z} \stackrel{\text{def}}{=} (\mathbf{z}_1 | \mathbf{z}_2 | \mathbf{z}_3 | \dots)$, and also $\mathbf{f} = (\mathbf{s} | \mathbf{e}_2 | \mathbf{e}_3 | \dots)$ then we get the LWE instance

$$\mathbf{z} = \mathbf{e}_1^t B + \mathbf{f}$$

with secret \mathbf{e}_1^t . The thing that makes this LWE instance potentially easier than the original one is that the first part of the error vector \mathbf{f} is our sparse/small vector \mathbf{s} , so the transformed instance has smaller error than the original (which means that it is easier to solve).

Trying to quantify the effect of this attack, we note that the optimal M value in the attack from Appendix C.1 above is obtained at $M = 2N$, which means that the new error vector is $\mathbf{f} = (\mathbf{s} | \mathbf{e}_2)$, which has Euclidean norm smaller than $\mathbf{e} = (\mathbf{e}_1 | \mathbf{e}_2)$ by roughly a factor of $\sqrt{2}$ (assuming that $\|\mathbf{s}\| \ll \|\mathbf{e}_1\| \approx \|\mathbf{e}_2\|$). Maybe some further improvement can be obtained by using a smaller value for M , where the shorter error may outweigh the “non optimal” value of M . However, we do not expect to get major improvement this way, so it seems that the very sparse secret should only add maybe one bit to the modulus/noise ratio.

C.2 The Modulus Size

In this section we assume that we are given the parameter $N = \phi(m)$ (for our polynomial ring modulo $\Phi_m(X)$). We also assume that we are given the noise variance σ^2 , the number of levels in the modulus chain L , an additional “slackness parameter” ξ (whose purpose is explained below), and the number of nonzero coefficients in the secret key h . Our goal is to devise a lower bound on the size of the largest modulus Q used in the public key, so as to maintain the functionality of the scheme.

Controlling the Noise. Driving the analysis in this section is a bound on the noise magnitude right after modulus switching, which we denote below by B . We set our parameters so that starting from ciphertexts with noise magnitude B , we can perform one level of fan-in-two multiplications, then one level of fan-in- ξ additions, followed by key switching and modulus switching again, and get the noise magnitude back to the same B .

- Recall that in the “reduced canonical embedding norm”, the noise magnitude is at most multiplied by modular multiplication and added by modular addition, hence after the multiplication and addition levels the noise magnitude grows from B to as much as ξB^2 .
- As we’ve seen in Appendix B.3, performing key switching scales up the noise magnitude by a factor of P and adds another noise term of magnitude upto $B_{\text{Ks}} \cdot q_t$ (before doing modulus switching to scale it back down). Hence starting from noise magnitude ξB^2 , the noise grows to magnitude $P\xi B^2 + B_{\text{Ks}} \cdot q_t$ (relative to the modulus Pq_t).

Below we assume that after key-switching we do modulus switching directly to a smaller modulus.

- After key-switching we can switch to the next modulus q_{t-1} to decrease the noise back to our bound B . Following the analysis from Appendix B.2, switching moduli from Q_t to q_{t-1} decreases the noise magnitude by a factor of $q_{t-1}/Q_t = 1/(P \cdot p_t)$, and then add a noise term of magnitude B_{scale} .

Starting from noise magnitude $P\xi B^2 + B_{\text{Ks}} \cdot q_t$ before modulus switching, the noise magnitude after modulus switching is therefore bounded whp by

$$\frac{P \cdot \xi B^2 + B_{\text{Ks}} \cdot q_t}{P \cdot p_t} + B_{\text{scale}} = \frac{\xi B^2}{p_t} + \frac{B_{\text{Ks}} \cdot q_{t-1}}{P} + B_{\text{scale}}$$

Using the analysis above, our goal next is to set the parameters B, P and the p_t ’s (as functions of N, σ, L, ξ and h) so that in every level t we get $\frac{\xi B^2}{p_t} + \frac{B_{\text{Ks}} \cdot q_{t-1}}{P} + B_{\text{scale}} \leq B$. Namely we need to satisfy at every level t the quadratic inequality (in B)

$$\frac{\xi}{p_t} B^2 - B + \underbrace{\left(\frac{B_{\text{Ks}} \cdot q_{t-1}}{P} + B_{\text{scale}} \right)}_{\text{denote this by } R_{t-1}} \leq 0. \quad (9)$$

Observe that (assuming that all the primes p_t are roughly the same size), it suffices to satisfy this inequality for the largest modulus $t = L - 2$, since R_{t-1} increases with larger t ’s. Noting that $R_{L-3} > B_{\text{scale}}$, we want to get this term to be as close to B_{scale} as possible, which we can do by setting P large enough. Specifically, to make it as close as $R_{L-3} = (1 + 2^{-n})B_{\text{scale}}$ it is sufficient to set

$$P \approx 2^n \frac{B_{\text{Ks}} q_{L-3}}{B_{\text{scale}}} \approx 2^n \frac{9\sigma N q_{L-3}}{77\sqrt{N}} \approx 2^{n-3} q_{L-3} \cdot \sigma \sqrt{N}, \quad (10)$$

Below we set (say) $n = 8$, which makes it close enough to use just $R_{L-3} \approx B_{\text{scale}}$ for the derivation below.

Clearly to satisfy Inequality (9) we must have a positive discriminant, which means $1 - 4\frac{\xi}{p_{L-2}} R_{L-3} \geq 0$, or $p_{L-2} \geq 4\xi R_{L-3}$. Using the value $R_{L-3} \approx B_{\text{scale}}$, this translates into setting

$$p_1 \approx p_2 \cdots \approx p_{L-2} \approx 4\xi \cdot B_{\text{scale}} \approx 308\xi \sqrt{N} \quad (11)$$

Finally, with the discriminant positive and all the p_i ’s roughly the same size we can satisfy Inequality (9) by setting

$$B \approx \frac{1}{2\xi/p_{L-2}} = \frac{p_{L-2}}{2\xi} \approx 2B_{\text{scale}} \approx 154\sqrt{N}. \quad (12)$$

The Smallest Modulus. After evaluating our L -level circuit, we arrive at the last modulus $q_0 = p_0$ with noise bounded by ξB^2 . To be able to decrypt, we need this noise to be smaller than $q_0/2c_m$, where c_m is the ring constant for our polynomial ring modulo $\Phi_m(X)$. For our setting, that constant is always below 40, so a sufficient condition for being able to decrypt is to set

$$q_0 = p_0 \approx 80\xi B^2 \approx 2^{20.9}\xi N \quad (13)$$

The Encryption Modulus. Recall that freshly encrypted ciphertext have noise B_{clean} (as defined in Equation (6)), which is larger than our baseline bound B from above. To reduce the noise magnitude after the first modulus switching down to B , we therefore set the ratio $p_{L-1} = q_{L-1}/q_{L-2}$ so that $B_{\text{clean}}/p_{L-1} + B_{\text{scale}} \leq B$. This means that we set

$$p_{L-1} = \frac{B_{\text{clean}}}{B - B_{\text{scale}}} \approx \frac{74N + 858\sqrt{N}}{77\sqrt{N}} \approx \sqrt{N} + 11 \quad (14)$$

The Largest Modulus. Having set all the parameters, we are now ready to calculate the resulting bound on the largest modulus, namely $Q_{L-2} = q_{L-2} \cdot P$. Using Equations (11), and (13), we get

$$q_t = p_0 \cdot \prod_{i=1}^t p_i \approx (2^{20.9}\xi N) \cdot (308\xi\sqrt{N})^t = 2^{20.9} \cdot 308^t \cdot \xi^{t+1} \cdot N^{t/2+1}. \quad (15)$$

Now using Equation (10) we have

$$\begin{aligned} P &\approx 2^5 q_{L-3} \sigma \sqrt{N} \approx 2^{25.9} \cdot 308^{L-3} \cdot \xi^{L-2} \cdot N^{(L-3)/2+1} \cdot \sigma \sqrt{N} \\ &\approx 2 \cdot 308^L \cdot \xi^{L-2} \sigma N^{L/2} \end{aligned}$$

and finally

$$\begin{aligned} Q_{L-2} = P \cdot q_{L-2} &\approx (2 \cdot 308^L \cdot \xi^{L-2} \sigma N^{L/2}) \cdot (2^{20.9} \cdot 308^{L-2} \cdot \xi^{L-1} \cdot N^{L/2}) \\ &\approx \sigma \cdot 2^{16.5L+5.4} \cdot \xi^{2L-3} \cdot N^L \end{aligned} \quad (16)$$

C.3 Putting It Together

We now have in Equation (8) a lower bound on N in terms of Q, σ and the security level k , and in Equation (16) a lower bound on Q with respect to N, σ and several other parameters. We note that σ is a free parameter, since it drops out when substituting Equation (16) in Equation (8). In our implementation we used $\sigma = 3.2$, which is the smallest value consistent with the analysis in [23].

For the other parameters, we set $\xi = 8$ (to get a small “wiggle room” without increasing the parameters much), and set the number of nonzero coefficients in the secret key at $h = 64$ (which is already included in the formulas from above, and should easily defeat exhaustive-search/birthday type of attacks). Substituting these values into the equations above we get

$$\begin{aligned} p_0 &\approx 2^{23.9}N, \quad p_i \approx 2^{11.3}\sqrt{N} \text{ for } i = 1, \dots, L-2 \\ P &\approx 2^{11.3L-5}N^{L/2}, \text{ and } Q_{L-2} \approx 2^{22.5L-3.6}\sigma N^L. \end{aligned}$$

Substituting the last value of Q_{L-2} into Equation (8) yields

$$N > \frac{(L(\log N + 23) - 8.5)(k + 110)}{7.2} \quad (17)$$

Targeting $k = 80$ -bits of security and solving for several different depth parameters L , we get the results in the table below, which also lists approximate sizes for the primes p_i and P .

L	N	$\log_2(p_0)$	$\log_2(p_i)$	$\log_2(p_{L-1})$	$\log_2(P)$
10	9326	37.1	17.9	7.5	177.3
20	19434	38.1	18.4	8.1	368.8
30	29749	38.7	18.7	8.4	564.2
40	40199	39.2	18.9	8.6	762.2
50	50748	39.5	19.1	8.7	962.1
60	61376	39.8	19.2	8.9	1163.5
70	72071	40.0	19.3	9.0	1366.1
80	82823	40.2	19.4	9.1	1569.8
90	93623	40.4	19.5	9.2	1774.5

Choosing Concrete Values. Having obtained lower-bounds on $N = \phi(m)$ and other parameters, we now need to fix precise cyclotomic fields $\mathbb{Q}(\zeta_m)$ to support the algebraic operations we need. We have two situations we will be interested in for our experiments. The first corresponds to performing arithmetic on bytes in \mathbb{F}_{2^8} (i.e. $n = 8$), whereas the latter corresponds to arithmetic on bits in \mathbb{F}_2 (i.e. $n = 1$). We therefore need to find an odd value of m , with $\phi(m) \approx N$ and m dividing $2^d - 1$, where we require that d is divisible by n . Values of m with a small number of prime factors are preferred as they give rise to smaller values of c_m . We also look for parameters which maximize the number of slots ℓ we can deal with in one go, and values for which $\phi(m)$ is close to the approximate value for N estimated above. When $n = 1$ we always select a set of parameters for which the ℓ value is at least as large as that obtained when $n = 8$.

L	$n = 8$				$n = 1$			
	m	$N = \phi(m)$	(d, ℓ)	c_K	m	$N = \phi(m)$	(d, ℓ)	c_K
10	11441	10752	(48,224)	3.60	11023	10800	(45,240)	5.13
20	34323	21504	(48,448)	6.93	34323	21504	(48,448)	6.93
30	31609	31104	(72,432)	5.15	32377	32376	(57,568)	1.27
40	54485	40960	(64,640)	12.40	42799	42336	(21,2016)	5.95
50	59527	51840	(72,720)	21.12	54161	52800	(60,880)	4.59
60	68561	62208	(72,864)	36.34	85865	63360	(60,1056)	12.61
70	82603	75264	(56,1344)	36.48	82603	75264	(56,1344)	36.48
80	92837	84672	(56,1512)	38.52	101437	85672	(42,2016)	19.13
90	124645	98304	(48,2048)	21.07	95281	94500	(45,2100)	6.22

D $\text{Scale}(c, q_t, q_{t-1})$ in dble-CRT Representation

Let $q_i = \prod_{j=0}^i p_j$, where the p_j 's are primes that split completely in our cyclotomic field \mathbb{A} . We are given a $c \in \mathbb{A}_{q_t}$ represented via double-CRT – that is, it is represented as a “matrix” of its evaluations at the primitive m -th roots of unity modulo the primes p_0, \dots, p_t . We want to modulus switch to q_{t-1} – i.e., scale

down by a factor of p_t . Let's recall what this means: we want to output $c' \in \mathbb{A}$, represented via double-CRT format (as its matrix of evaluations modulo the primes p_0, \dots, p_{t-1}), such that

1. $c' = c \bmod 2$.
2. c' is very close (in terms of its coefficient vector) to c/p_t .

In the main body we explained how this could be performed in dble-CRT representation. This made explicit use of the fact that the two ciphertexts need to be equivalent modulo two. If we wished to replace two with a general prime p , then things are a bit more complicated. For completeness, although it is not required in our scheme, we present a methodology below. In this case, the conditions on c^\dagger are as follows:

1. $c^\dagger = c \cdot p_t \bmod p$.
2. c^\dagger is very close to c .
3. c^\dagger is divisible by p_t .

As before, we set $c' \leftarrow c^\dagger/p_t$. (Note that for $p = 2$, we trivially have $c \cdot p_t = c \bmod p$, since p_t will be odd.)

This causes some complications, because we set $c^\dagger \leftarrow c + \delta$, where $\delta = -\bar{c} \bmod p_t$ (as before) but now $\delta = (p_t - 1) \cdot c \bmod p$. To compute such a δ , we need to know $c \bmod p$. Unfortunately, we don't have $c \bmod p$. One not-very-satisfying way of dealing with this problem is the following. Set $\hat{c} \leftarrow [p_t]_p \cdot c \bmod q_t$. Now, if c encrypted m , then \hat{c} encrypts $[p_t]_p \cdot m$, and \hat{c} 's noise is $[p_t]_p < p/2$ times as large. It is obviously easy to compute \hat{c} 's double-CRT format from c 's. Now, we set c^\dagger so that the following is true:

1. $c^\dagger = \hat{c} \bmod p$.
2. c^\dagger is very close to \hat{c} .
3. c^\dagger is divisible by p_t .

This is easy to do. The algorithm to output c^\dagger in double-CRT format is as follows:

1. Set \bar{c} to be the coefficient representation of $\hat{c} \bmod p_t$. (Computing this requires a single "small FFT" modulo the prime p_t .)
2. Set δ to be the polynomial with coefficients in $(-p_t \cdot p/2, p_t \cdot p/2]$ such that $\delta = 0 \bmod p$ and $\delta = -\bar{c} \bmod p_t$.
3. Set $c^\dagger = \hat{c} + \delta$, and output c^\dagger 's double-CRT representation.
 - (a) We already have \hat{c} 's double-CRT representation.
 - (b) Computing δ 's double-CRT representation requires t "small FFTs" modulo the p_j 's.

E Other Optimizations

Some other optimizations that we encountered during our implementation work are discussed next. Not all of these optimizations are useful for our current implementation, but they may be useful in other contexts.

Three-way Multiplications. Sometime we need to multiply several ciphertexts together, and if their number is not a power of two then we do not have a complete binary tree of multiplications, which means that at some point in the process we will have three ciphertexts that we need to multiply together.

The standard way of implementing this 3-way multiplication is via two 2-argument multiplications, e.g., $x \cdot (y \cdot z)$. But it turns out that here it is better to use “raw multiplication” to multiply these three ciphertexts (as done in [7]), thus getting an “extended” ciphertext with four elements, then apply key-switching (and later modulus switching) to this ciphertext. This takes only six ring-multiplication operations (as opposed to eight according to the standard approach), three modulus switching (as opposed to four), and only one key switching (applied to this 4-element ciphertext) rather than two (which are applied to 3-element extended ciphertexts). All in all, this three-way multiplication takes roughly 1.5 times a standard two-element multiplication.

We stress that this technique is not useful for larger products, since for more than three multiplicands the noise begins to grow too large. But with only three multiplicands we get noise of roughly B^3 after the multiplication, which can be reduced to noise $\approx B$ by dropping two levels, and this is also what we get by using two standard two-element multiplications.

Commuting Automorphisms and Multiplications. Recalling that the automorphisms $X \mapsto X^i$ commute with the arithmetic operations, we note that some ordering of these operations can sometimes be better than others. For example, it may be better perform the multiplication-by-constant before the automorphism operation whenever possible. The reason is that if we perform the multiply-by-constant after the key-switching that follows the automorphism, then added noise term due to that key-switching is multiplied by the same constant, thereby making the noise slightly larger. We note that to move the multiplication-by-constant before the automorphism, we need to multiply by a different constant.

Switching to higher-level moduli. We note that it may be better to perform automorphisms at a higher level, in order to make the added noise term due to key-switching small with respect to the modulus. On the other hand operations at high levels are more expensive than the same operations at a lower level. A good rule of thumb is to perform the automorphism operations one level above the lowest one. Namely, if the naive strategy that never switches to higher-level moduli would perform some Frobenius operation at level q_i , then we perform the key-switching following this Frobenius operation at level Q_{i+1} , and then switch back to level q_{i+1} (rather than using Q_i and q_i).

Commuting Addition and Modulus-switching. When we need to add many terms that were obtained from earlier operations (and their subsequent key-switching), it may be better to first add all of these terms relative to the large modulus Q_i before switching the sum down to the smaller q_i (as opposed to switching all the terms individually to q_i and then adding).

Reducing the number of key-switching matrices. When using many different automorphisms $\kappa_i : X \mapsto X^i$ we need to keep many different key-switching matrices in the public key, one for every value of i that we use. We can reduce this memory requirement, at the expense of taking longer to perform the automorphisms. We use the fact that the Galois group \mathcal{G} that contains all the maps κ_i (which is isomorphic to $(\mathbb{Z}/m\mathbb{Z})^*$) is generated by a relatively small number of generators. (Specifically, for our choice of parameters the group $(\mathbb{Z}/m\mathbb{Z})^*$ has two or three generators.) It is therefore enough to store in the public key only the key-switching matrices corresponding to κ_{g_j} ’s for these generators g_j of the group \mathcal{G} . Then in order

to apply a map κ_i we express it as a product of the generators and apply these generators to get the effect of κ_i . (For example, if $i = g_1^2 \cdot g_2$ then we need to apply κ_{g_1} twice followed by a single application of κ_{g_2} .)

Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP

Zvika Brakerski*

Abstract

We present a new tensoring technique for LWE-based fully homomorphic encryption. While in all previous works, the ciphertext noise grows quadratically ($B \rightarrow B^2 \cdot \text{poly}(n)$) with every multiplication (before “refreshing”), our noise only grows linearly ($B \rightarrow B \cdot \text{poly}(n)$).

We use this technique to construct a *scale-invariant* fully homomorphic encryption scheme, whose properties only depend on the ratio between the modulus q and the initial noise level B , and not on their absolute values.

Our scheme has a number of advantages over previous candidates: It uses the same modulus throughout the evaluation process (no need for “modulus switching”), and this modulus can take arbitrary form. In addition, security can be *classically* reduced from the worst-case hardness of the GapSVP problem (with quasi-polynomial approximation factor), whereas previous constructions could only exhibit a quantum reduction from GapSVP.

*Stanford University, zvika@stanford.edu. Supported by a Simons Postdoctoral Fellowship and by DARPA.

1 Introduction

Fully homomorphic encryption has been the focus of extensive study since the first candidate scheme was introduced by Gentry [Gen09b]. In a nutshell, fully homomorphic encryption allows to perform arbitrary computation on encrypted data. It can thus be used, for example, to outsource a computation to a remote server without compromising data privacy.

The first generation of fully homomorphic schemes [Gen09b, DGHV10, SV10, BV11a, CMNT11, GH11] that started with Gentry’s seminal work, all followed a similar and fairly complicated methodology, often relying on relatively strong computational assumptions. A second generation of schemes started with the work of Brakerski and Vaikuntanathan [BV11b], who established full homomorphism in a simpler way, based on the *learning with errors* (LWE) assumption. Using known reductions [Reg05, Pei09], the security of their construction is based on the (often quantum) hardness of approximating some short vector problems in worst-case lattices. Their scheme was then improved by Brakerski, Gentry and Vaikuntanathan [BGV12], as we describe below.

In LWE-based schemes such as [BV11b, BGV12], ciphertexts are represented as vectors in \mathbb{Z}_q , for some modulus q . The decryption process is essentially computing an inner product of the ciphertext and the secret key vector, which produces a noisy version of the message (the noise is added at encryption for security purposes). The noise increases with every homomorphic operation, and correct decryption is guaranteed if the final noise magnitude is below $q/4$. Homomorphic addition roughly doubles the noise, while homomorphic multiplication roughly squares it.

In the [BV11b] scheme, after L levels of multiplication (e.g. evaluating a depth L multiplication tree), the noise grows from an initial magnitude of B , to B^{2^L} . Hence, to enable decryption, a very large modulus $q \approx B^{2^L}$ was required. This affected both efficiency and security (the security of the scheme depends inversely on the ratio q/B , so bigger q for the same B means less security).

The above was improved by [BGV12], who suggested to *scale down* the ciphertext vector after every multiplication (they call this “modulus switching”, see below).¹ That is, to go from a vector \mathbf{c} over \mathbb{Z}_q , into the vector \mathbf{c}/w over $\mathbb{Z}_{q/w}$ (for some scaling factor w). Scaling “switches” the modulus q to a smaller q/w , but also reduces the noise by the same factor (from B to B/w). To see why this change of scale is effective, consider scaling by a factor B after every multiplication (as indeed suggested by [BGV12]): After the first multiplication, the noise goes up to B^2 , but scaling brings it back down to B , at the cost of reducing the modulus to q/B . With more multiplications, the noise magnitude always goes back to B , but the modulus keeps reducing. After L levels of multiplication-and-scaling, the noise magnitude is still B , but the modulus is down to q/B^L . Therefore it is sufficient to use $q \approx B^{L+1}$, which is significantly lower than before. However, this process results in a complicated homomorphic evaluation process that “climbs down the ladder of moduli”.

The success of the scaling methodology teaches us that *perspective* matters: scaling does not change the ratio between the modulus and noise, but it still manages the noise better by changing the perspective in which we view the ciphertext. In this work, we suggest to work in an *invariant perspective* where only the ratio q/B matters (and not the absolute values of q, B as in previous works). We derive a scheme that is superior to the previous best known in simplicity, noise management and security. Details follow.

¹A different scaling technique was already suggested in [BV11b] as a way to simplify decryption and improve efficiency, but not to manage noise.

1.1 Our Results

As explained above, we present a *scale invariant* scheme, by finding an invariant perspective. The idea is very natural based on the outlined motivation: if we scale down the ciphertext by a factor of q , we get a fractional ciphertext modulo 1, with noise magnitude B/q . In this perspective, all choices of q, B with the same B/q ratio will look the same. It turns out that in this perspective, homomorphic multiplication does not square the noise, but rather multiplies it by a polynomial factor $p(n)$ that depends only on the security parameter.² After L levels of multiplication, the noise will grow from B/q to $(B/q) \cdot p(n)^L$, which means that we only need to use $q \approx B \cdot p(n)^L$.

Interestingly, the idea of working modulo 1 goes back to the early works of Ajtai and Dwork [AD97], and Regev [Reg03], and to the first formulation of LWE [Reg05]. In a sense, we are “going back to the roots” and showing that these early ideas are instrumental in the construction of homomorphic encryption.

For technical reasons, we don’t implement the scheme over fractions, but rather mimic the invariant perspective over \mathbb{Z}_q (see Section 1.2 for more details). Perhaps surprisingly, the resulting scheme is exactly Regev’s original LWE-based scheme, with additional auxiliary information for the purpose of homomorphic evaluation. The properties of our scheme are summarized in the following theorem:

Theorem. *There exists a homomorphic encryption scheme for depth L circuits, based on the $\text{DLWE}_{n,q,\chi}$ assumption (n -dimensional decision-LWE modulo q , with noise χ), so long as*

$$q/B \geq (O(n \log q))^{L+O(1)},$$

where B is a bound on the values of χ .

The resulting scheme has a number of interesting properties:

1. **Scale invariance.** Homomorphic properties only depend on q/B (as explained above).
2. **No modulus switching.** We work with a single modulus q . We don’t need to switch moduli as in [BV11b, BGV12]. This leads to a simpler description of the scheme (and hopefully better implementations).
3. **No restrictions on the modulus.** Our modulus q can take any form (so long as it satisfies the size requirement). This is achieved by putting the message bit in the most significant bit of the ciphertext, rather than least significant as in previous homomorphic schemes (this can be interpreted as making the *message* scale invariant). We note that for odd q , the least and most significant bit representations are interchangeable.

In particular, in our scheme q can be a power of 2, which can simplify implementation of arithmetics.³ In previous schemes, such q could not be used for binary message spaces.⁴

This, again, is going back to the roots: Early schemes such as [Reg05], and in a sense also [AD97, GGH97], encoded ciphertexts in the most significant bits. Switching to least significant bit encoding was (perhaps ironically) motivated by improving homomorphism.

²More accurately, a polynomial $p(n, \log q)$, but w.l.o.g $q \leq 2^n$.

³On the downside, such q might reduce efficiency when using ring-LWE (see below) due to FFT embedding issues.

⁴[GHS11a] gain on efficiency by using moduli that are “almost” a power of 2.

4. **No restrictions on the secret key distribution.** While [BGV12] requires that the secret key is drawn from the noise distribution (LWE in Hermite normal form), our scheme works under any secret key distribution for which the LWE assumption holds.
5. **Classical Reduction from GapSVP.** One of the appeals of LWE-based cryptography is the known quantum (Regev [Reg05]) and classical (Peikert [Pei09]) reductions from the worst case hardness of lattice problems. Specifically to GapSVP_γ , which is the problem of deciding, given an n dimensional lattice and a number d , between the following two cases: either the lattice has a vector shorter than d , or it doesn't have any vector shorter than $\gamma(n) \cdot d$. The value of γ depends on the ratio q/B (essentially $\gamma = (q/B) \cdot \tilde{O}(n)$), and the smaller γ is, the better the reduction ($\text{GapSVP}_{2^{\Omega(n)}}$ is an easy problem).

Peikert's classical reduction requires that $q \approx 2^{n/2}$, which makes his reduction unusable for previous homomorphic schemes, since γ becomes exponential. For example, in [BGV12], $q/B = q/q^{1/(L+1)} = q^{1-1/(L+1)}$ which translates to $\gamma \approx 2^{n/2}$ for the required q .⁵

In our scheme, this problem does not arise. We can instantiate our scheme with any q while hardly affecting the ratio q/B . We can therefore set $q \approx 2^{n/2}$ and get a classical reduction from $\text{GapSVP}_{n^{O(\log n)}}$, which is currently solvable only in $2^{\tilde{\Omega}(n)}$ time. (This is mostly of theoretical interest, though, since efficiency considerations will favor the smallest possible q .)

Using our scheme as a building block we achieve:

1. **Fully homomorphic encryption using bootstrapping.** Using Gentry's bootstrapping theorem, we present a leveled fully homomorphic scheme based on the classical worst case $\text{GapSVP}_{n^{O(\log n)}}$ problem. As usual, an additional circular security assumption is required to get a non-leveled scheme.
2. **Leveled fully homomorphic encryption without bootstrapping.** Very similarly to [BGV12], our scheme can be used to achieve leveled homomorphism without bootstrapping.
3. **Increased efficiency using ring-LWE (RLWE).** RLWE (defined in [LPR10]) is a version of LWE that works over polynomial rings rather than the integers. Its hardness is quantumly related to short vector problems in *ideal* lattices.

RLWE is a stronger assumption than LWE, but it can dramatically improve the efficiency of schemes [BV11a, BGV12, GHS11b]. Our methods are readily portable to the RLWE world.

In summary, our construction carries conceptual significance in its simplicity and in a number of theoretical aspects. Its practical usefulness compared to other schemes is harder to quantify, though, since it will vary greatly with the specific implementation and optimizations chosen.

1.2 Our Techniques

Our starting point is Regev's public key encryption scheme. There, the encryption of a message $m \in \{0, 1\}$ is an integer vector \mathbf{c} such that $\langle \mathbf{c}, \mathbf{s} \rangle = \lfloor \frac{q}{2} \rfloor \cdot m + e + qI$, for an integer I and for $|e| \leq E$, for some bound $E < q/4$. The secret key vector \mathbf{s} is also over the integers. We can assume w.l.o.g

⁵Peikert suggests to classically base small- q LWE on a new lattice problem that he introduces.

that the elements of \mathbf{c}, \mathbf{s} are in the segment $(-q/2, q/2]$. (We note that previous homomorphic constructions used a different variant of Regev's scheme, where $\langle \mathbf{c}, \mathbf{s} \rangle = m + 2e + qI$.)

In this work, we take the invariant perspective on the scheme, and consider the fractional ciphertext $\tilde{\mathbf{c}} = \mathbf{c}/q$. It holds that $\langle \tilde{\mathbf{c}}, \mathbf{s} \rangle = \frac{1}{2} \cdot m + \tilde{e} + I$, where $I \in \mathbb{Z}$ and $|\tilde{e}| \leq E/q = \epsilon$. The elements of \mathbf{c} are now rational numbers in $(-1/2, 1/2]$. Note that the secret key does not change and is still over \mathbb{Z} .

Additive homomorphism is immediate: if \mathbf{c}_1 encrypts m_1 and \mathbf{c}_2 encrypts m_2 , then $\mathbf{c}_{\text{add}} = \mathbf{c}_1 + \mathbf{c}_2$ encrypts $[m_1 + m_2]_2$. The noise grows from ϵ to $\approx 2\epsilon$. Multiplicative homomorphism is achieved by *tensoring* the input ciphertexts:

$$\mathbf{c}_{\text{mult}} = 2 \cdot \mathbf{c}_1 \otimes \mathbf{c}_2 .$$

The tensored ciphertext can be decrypted using a tensored secret key because

$$\langle \underbrace{2 \cdot \mathbf{c}_1 \otimes \mathbf{c}_2}_{\mathbf{c}_{\text{mult}}}, \mathbf{s} \otimes \mathbf{s} \rangle = 2 \cdot \langle \mathbf{c}_1, \mathbf{s} \rangle \cdot \langle \mathbf{c}_2, \mathbf{s} \rangle .$$

A “key switching” mechanism developed in [BV11b] and generalized in [BGV12] allows to switch back from a tensored secret key into a “normal” one without much additional noise. The details of this mechanism are immaterial for this discussion. We focus on the noise growth in the tensored ciphertext.

We want to show that $2 \cdot \langle \mathbf{c}_1, \mathbf{s} \rangle \cdot \langle \mathbf{c}_2, \mathbf{s} \rangle \approx \frac{1}{2}m_1m_2 + e' + I'$, for a small e' . To do this, we let $I_1, I_2 \in \mathbb{Z}$ be integers such that $\langle \mathbf{c}_1, \mathbf{s} \rangle = \frac{1}{2}m_1 + e_1 + I_1$, and likewise for \mathbf{c}_2 . It can be verified that $|I_1|, |I_2|$ are bounded by $\approx \|\mathbf{s}\|_1$. We therefore get:

$$\begin{aligned} 2 \cdot \langle \mathbf{c}_1, \mathbf{s} \rangle \cdot \langle \mathbf{c}_2, \mathbf{s} \rangle &= 2 \cdot (\tfrac{1}{2}m_1 + e_1 + I_1) \cdot (\tfrac{1}{2}m_2 + e_2 + I_2) \\ &= \tfrac{1}{2}m_1m_2 + 2(e_1I_2 + e_2I_1) + e_1m_2 + e_2m_1 + 2e_1e_2 + \underbrace{(m_1I_2 + m_2I_1 + 2I_1I_2)}_{\in \mathbb{Z}} . \end{aligned}$$

Interestingly, the cross-term e_1e_2 that was responsible for the squaring of the noise in previous schemes, is now practically insignificant since $\epsilon^2 \ll \epsilon$. The significant noise term in the above expression is $2(e_1I_2 + e_2I_1)$, which is bounded by $O(\|\mathbf{s}\|_1) \cdot \epsilon$. All that is left to show now is that $\|\mathbf{s}\|_1$ is independent of B, q and only depends on n (recall that we allow dependence on $\log q \leq n$).

On the face of it, $\|\mathbf{s}\|_1 \approx n \cdot q$, since the elements of \mathbf{s} are integers in the segment $(-q/2, q/2]$. In order to reduce the norm, we use *binary decomposition* (which was used in [BV11b, BGV12] for different purposes). Let $\mathbf{s}^{(j)}$ denote the binary vector that contains the j^{th} bit from each element of \mathbf{s} . Namely $\mathbf{s} = \sum_j 2^j \mathbf{s}^{(j)}$. Then

$$\langle \mathbf{c}, \mathbf{s} \rangle = \sum_j 2^j \langle \mathbf{c}, \mathbf{s}^{(j)} \rangle = \langle (\mathbf{c}, 2\mathbf{c}, \dots), (\mathbf{s}^{(0)}, \mathbf{s}^{(1)}, \dots) \rangle .$$

This means that we can convert a ciphertext \mathbf{c} that corresponds to a secret key \mathbf{s} in \mathbb{Z} , into a modified ciphertext $(\mathbf{c}, 2\mathbf{c}, \dots)$ that corresponds to a *binary* secret key $(\mathbf{s}^{(0)}, \mathbf{s}^{(1)}, \dots)$. The norm of the binary key is at most its dimension, which is polynomial in n as required.⁶

⁶Reducing the norm of \mathbf{s} was also an issue in [BGV12]. There it was resolved by using LWE in Hermite normal form, where \mathbf{s} is sampled from the noise distribution and thus $\|\mathbf{s}\|_1 \approx n \cdot B$. This suffices when B must be very small, as in [BGV12], but not in our setting.

We point out that an alternative solution to the norm problem follows by using the dual-Regev scheme of [GPV08] as the basic building block. There, the secret key is natively binary and of low norm. (In addition, as noticed in previous works, working with dual-Regev naturally implies a weak form of homomorphic identity based encryption.) However, the ciphertexts and some other parameters will need to grow.

Finally, working with fractional ciphertexts brings about issues of precision in representation and other problems. We thus implement our scheme over \mathbb{Z} with appropriate scaling: Each rational number x in the above description will be represented by the integer $y = \lfloor qx \rfloor$ (which determines x up to an additive factor of $1/(2q)$). The addition operation $x_1 + x_2$ is mimicked by $y_1 + y_2 \approx \lfloor q(x_1 + x_2) \rfloor$. To mimic multiplication, we take $\lfloor (y_1 \cdot y_2)/q \rfloor \approx \lfloor x_1 \cdot x_2 \cdot q \rfloor$. Our tensored ciphertext for multiplication will thus be defined as $\lfloor \frac{2}{q} \cdot \mathbf{c}_1 \otimes \mathbf{c}_2 \rfloor$, where $\mathbf{c}_1, \mathbf{c}_2$ are integer vectors and the tensoring operation is over the integers. In this representation, encryption and decryption become identical to Regev's original scheme.

1.3 Paper Organization

Section 2 defines notational conventions (we define \mathbb{Z}_q in a slightly unconventional way, the reader is advised to take notice), introduces the LWE assumption and defines homomorphic encryption and related terms. Section 3 introduces our building blocks: Regev's encryption scheme, binary decomposition of vectors and the key switching mechanism. Finally, in Section 4 we present and analyze our scheme, and discuss several possible optimizations.

2 Preliminaries

For an integer q , we define the set $\mathbb{Z}_q \triangleq (-q/2, q/2] \cap \mathbb{Z}$. We stress that in this work, \mathbb{Z}_q is *not synonymous* with the ring $\mathbb{Z}/q\mathbb{Z}$. In particular, all arithmetics is performed over \mathbb{Z} (or \mathbb{Q} when division is used) and not over any sub-ring. For any $x \in \mathbb{Q}$, we let $y = [x]_q$ denote the unique value $y \in (-q/2, q/2]$ such that $y = x \pmod{q}$ (i.e. y is congruent to x modulo q).⁷

We use $\lfloor x \rfloor$ to indicate rounding x to the nearest integer, and $\lfloor x \rfloor, \lceil x \rceil$ (for $x \geq 0$) to indicate rounding down or up. All logarithms are to base 2.

Probability. We use $x \xleftarrow{\$} \mathcal{D}$ to denote that x is sampled from a distribution \mathcal{D} . Similarly, $x \xleftarrow{\$} S$ denotes that x is uniform over a set S . We define B -bounded distributions as ones whose magnitudes never exceed B .⁸

Definition 2.1. A distribution χ over the integers is B -bounded (denoted $|\chi| \leq B$) if it is only supported on $[-B, B]$.

A function is negligible if it vanishes faster than any inverse polynomial. Two distributions are statistically indistinguishable if the total variation distance between them is negligible, and computationally indistinguishable if no polynomial test distinguishes them with non-negligible advantage.

⁷For example, if $x = 2, y = -3 \in \mathbb{Z}_7$, then $x \cdot y = -6 \notin \mathbb{Z}_7$, however $[x \cdot y]_7 = 1 \in \mathbb{Z}_7$.

⁸This definition is simpler and slightly different from previous works.

Vectors, Matrices and Tensors. We denote scalars in plain (e.g. x) and vectors in bold lowercase (e.g. \mathbf{v}), and matrices in bold uppercase (e.g. \mathbf{A}). For the sake of brevity, we use (\mathbf{x}, \mathbf{y}) to refer to the vector $[\mathbf{x}^T \parallel \mathbf{y}^T]^T$.

The ℓ_i norm of a vector is denoted by $\|\mathbf{v}\|_i$. Inner product is denoted by $\langle \mathbf{v}, \mathbf{u} \rangle$, recall that $\langle \mathbf{v}, \mathbf{u} \rangle = \mathbf{v}^T \cdot \mathbf{u}$. Let \mathbf{v} be an n dimensional vector. For all $i = 1, \dots, n$, the i^{th} element in \mathbf{v} is denoted $\mathbf{v}[i]$. When applied to vectors, operators such as $[\cdot]_q, \lfloor \cdot \rfloor$ are applied element-wise.

The tensor product of two vectors \mathbf{v}, \mathbf{w} of dimension n , denoted $\mathbf{v} \otimes \mathbf{w}$, is the n^2 dimensional vector containing all elements of the form $\mathbf{v}[i]\mathbf{w}[j]$. Note that

$$\langle \mathbf{v} \otimes \mathbf{w}, \mathbf{x} \otimes \mathbf{y} \rangle = \langle \mathbf{v}, \mathbf{x} \rangle \cdot \langle \mathbf{w}, \mathbf{y} \rangle.$$

2.1 Learning With Errors (LWE)

The LWE problem was introduced by Regev [Reg05] as a generalization of “learning parity with noise”. For positive integers n and $q \geq 2$, a vector $\mathbf{s} \in \mathbb{Z}_q^n$, and a probability distribution χ on \mathbb{Z} , let $A_{\mathbf{s}, \chi}$ be the distribution obtained by choosing a vector $\mathbf{a} \xleftarrow{\$} \mathbb{Z}_q^n$ uniformly at random and a noise term $e \xleftarrow{\$} \chi$, and outputting $(\mathbf{a}, [\langle \mathbf{a}, \mathbf{s} \rangle + e]_q) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$. Decisional LWE (DLWE) is defined as follows.

Definition 2.2 (DLWE). *For an integer $q = q(n)$ and an error distribution $\chi = \chi(n)$ over \mathbb{Z} , the (average-case) decision learning with errors problem, denoted $\text{DLWE}_{n,m,q,\chi}$, is to distinguish (with non-negligible advantage) m samples chosen according to $A_{\mathbf{s}, \chi}$ (for uniformly random $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^n$), from m samples chosen according to the uniform distribution over $\mathbb{Z}_q^n \times \mathbb{Z}_q$. We denote by $\text{DLWE}_{n,q,\chi}$ the variant where the adversary gets oracle access to $A_{\mathbf{s}, \chi}$, and is not a-priori bounded in the number of samples.*

There are known quantum (Regev [Reg05]) and classical (Peikert [Pei09]) reductions between $\text{DLWE}_{n,m,q,\chi}$ and approximating short vector problems in lattices. Specifically, these reductions take χ to be (discretized versions of) the Gaussian distribution, which is statistically indistinguishable from B -bounded, for an appropriate B . Since the exact distribution χ does not matter for our results, we state a corollary of the results of [Reg05, Pei09] (in conjunction with the search to decision reduction of Micciancio and Mol [MM11] and Micciancio and Peikert [MP11]) in terms of the bound B . These results also extend to additional forms of q (see [MM11, MP11]).

Corollary 2.1 ([Reg05, Pei09, MM11, MP11]). *Let $q = q(n) \in \mathbb{N}$ be either a prime power $q = p^r$, or a product of co-prime numbers $q = \prod q_i$ such that for all i , $q_i = \text{poly}(n)$, and let $B \geq \omega(\log n) \cdot \sqrt{n}$. Then there exists an efficiently sampleable B -bounded distribution χ such that if there is an efficient algorithm that solves the (average-case) $\text{DLWE}_{n,q,\chi}$ problem. Then:*

- *There is an efficient quantum algorithm that solves $\text{GapSVP}_{\tilde{O}(n \cdot q/B)}$ (and $\text{SIVP}_{\tilde{O}(n \cdot q/B)}$) on any n -dimensional lattice.*
- *If in addition $q \geq \tilde{O}(2^{n/2})$, then there is an efficient classical algorithm for $\text{GapSVP}_{\tilde{O}(n \cdot q/B)}$ on any n -dimensional lattice.*

In both cases, if one also considers distinguishers with sub-polynomial advantage, then we require $B \geq \tilde{O}(n)$ and the resulting approximation factor is slightly larger $\tilde{O}(n\sqrt{n} \cdot q/B)$.

Recall that GapSVP_γ is the (promise) problem of distinguishing, given a basis for a lattice and a parameter d , between the case where the lattice has a vector shorter than d , and the case where the lattice doesn't have any vector shorter than $\gamma \cdot d$. SIVP is the search problem of finding a set of “short” vectors. We refer the reader to [Reg05, Pei09] for more information.

The best known algorithms for GapSVP_γ ([Sch87, MV10]) require at least $2^{\tilde{\Omega}(n/\log \gamma)}$ time. The scheme we present in this work reduces from $\gamma = n^{O(\log n)}$, for which the best known algorithms run in time $2^{\tilde{\Omega}(n)}$.

As a final remark, we mention that Peikert also shows a classical reduction in the case of small values of q , but this reduction is from a newly defined “ ζ -to- γ decisional shortest vector problem”, which is not as extensively studied as GapSVP .

2.2 Homomorphic Encryption and Bootstrapping

We now define homomorphic encryption and introduce Gentry's bootstrapping theorem. Our definitions are mostly taken from [BV11b, BGV12].

A homomorphic (public-key) encryption scheme $\text{HE} = (\text{HE.Keygen}, \text{HE.Enc}, \text{HE.Dec}, \text{HE.Eval})$ is a quadruple of PPT algorithms as follows (n is the security parameter):

- **Key generation** $(pk, evk, sk) \leftarrow \text{HE.Keygen}(1^n)$: Outputs a public encryption key pk , a public evaluation key evk and a secret decryption key sk .⁹
- **Encryption** $c \leftarrow \text{HE.Enc}_{pk}(m)$: Using the public key pk , encrypts a single bit message $m \in \{0, 1\}$ into a ciphertext c .
- **Decryption** $m \leftarrow \text{HE.Dec}_{sk}(c)$: Using the secret key sk , decrypts a ciphertext c to recover the message $m \in \{0, 1\}$.
- **Homomorphic evaluation** $c_f \leftarrow \text{HE.Eval}_{evk}(f, c_1, \dots, c_\ell)$: Using the evaluation key evk , applies a function $f : \{0, 1\}^\ell \rightarrow \{0, 1\}$ to c_1, \dots, c_ℓ , and outputs a ciphertext c_f .

As in previous works, we represent f as an arithmetic circuit over $\text{GF}(2)$ with addition and multiplication gates. Thus it is customary to “break” HE.Eval into homomorphic addition $c_{\text{add}} \leftarrow \text{HE.Add}_{evk}(c_1, c_2)$ and homomorphic multiplication $c_{\text{mult}} \leftarrow \text{HE.Mult}_{evk}(c_1, c_2)$.

A homomorphic encryption scheme is said to be secure if it is semantically secure (note that the adversary is given both pk and evk).

Homomorphism w.r.t depth-bounded circuits and full homomorphism are defined next:

Definition 2.3 (L -homomorphism). *A scheme HE is L -homomorphic, for $L = L(n)$, if for any depth L arithmetic circuit f (over $\text{GF}(2)$) and any set of inputs m_1, \dots, m_ℓ , it holds that*

$$\Pr [\text{HE.Dec}_{sk}(\text{HE.Eval}_{evk}(f, c_1, \dots, c_\ell)) \neq f(m_1, \dots, m_\ell)] = \text{negl}(n),$$

where $(pk, evk, sk) \leftarrow \text{HE.Keygen}(1^n)$ and $c_i \leftarrow \text{HE.Enc}_{pk}(m_i)$.

Definition 2.4 (compactness and full homomorphism). *A homomorphic scheme is compact if its decryption circuit is independent of the evaluated function. A compact scheme is fully homomorphic if it is L -homomorphic for any polynomial L . The scheme is leveled fully homomorphic if it takes 1^L as additional input in key generation.*

⁹We adopt the terminology of [BV11b] that treats the evaluation key as a separate entity from the public key.

Gentry's bootstrapping theorem shows how to go from L -homomorphism to full homomorphism:

Theorem 2.2 (bootstrapping [Gen09b, Gen09a]). *If there exists an L -homomorphic scheme whose decryption circuit depth is less than L , then there exists a leveled fully homomorphic encryption scheme.*

Furthermore, if the aforementioned L -homomorphic scheme is also weak circular secure (remains secure even against an adversary who gets encryptions of the bits of the secret key), then there exists a fully homomorphic encryption scheme.

3 Building Blocks

In this section, we present building blocks from previous works that are used in our construction. Specifically, like all LWE-based fully homomorphic schemes, we rely on Regev's [Reg05] basic public-key encryption scheme (Section 3.1). We also use the key-switching methodology of [BV11b, BGV12] (Section 3.2).

3.1 Regev's Encryption Scheme

Let $q = q(n)$ be an integer function and let $\chi = \chi(n)$ be a distribution ensemble over \mathbb{Z} . The scheme Regev is defined as follows:

- **Regev.SecretKeygen**(1^n): Sample $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^n$. Output $sk = \mathbf{s}$.
- **Regev.PublicKeygen**(\mathbf{s}): Let $N \triangleq (n + 1) \cdot (\log q + O(1))$. Sample $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{N \times n}$ and $\mathbf{e} \xleftarrow{\$} \chi^N$. Compute $\mathbf{b} := [\mathbf{A} \cdot \mathbf{s} + \mathbf{e}]_q$, and define

$$\mathbf{P} := [\mathbf{b} \parallel -\mathbf{A}] \in \mathbb{Z}_q^{N \times (n+1)}.$$

Output $pk = \mathbf{P}$.

- **Regev.Enc** _{pk} (m): To encrypt a message $m \in \{0, 1\}$ using $pk = \mathbf{P}$, sample $\mathbf{r} \in \{0, 1\}^N$ and output ciphertext

$$\mathbf{c} := \left[\mathbf{P}^T \cdot \mathbf{r} + \left\lfloor \frac{q}{2} \right\rfloor \cdot \mathbf{m} \right]_q \in \mathbb{Z}_q^{n+1},$$

where $\mathbf{m} \triangleq (m, 0, \dots, 0) \in \{0, 1\}^{n+1}$.

- **Regev.Dec** _{sk} (\mathbf{c}): To decrypt $\mathbf{c} \in \mathbb{Z}_q^{n+1}$ using secret key $sk = \mathbf{s}$, compute

$$m := \left\lfloor \left[2 \cdot \frac{[\langle \mathbf{c}, (1, \mathbf{s}) \rangle]_q}{q} \right] \right\rfloor_2.$$

Correctness. We analyze the noise magnitude at encryption and decryption. We start with a lemma regarding the noise magnitude of properly encrypted ciphertexts:

Lemma 3.1 (encryption noise). *Let $q, n, N, |\chi| \leq B$ be parameters for Regev. Let $\mathbf{s} \in \mathbb{Z}^n$ be any vector and $m \in \{0, 1\}$ be some bit. Set $\mathbf{P} \leftarrow \text{Regev.PublicKeygen}(\mathbf{s})$ and $\mathbf{c} \leftarrow \text{Regev.Enc}_{\mathbf{P}}(m)$. Then for some e with $|e| \leq N \cdot B$ it holds that*

$$\langle \mathbf{c}, (1, \mathbf{s}) \rangle = \left\lfloor \frac{q}{2} \right\rfloor \cdot m + e \pmod{q}.$$

Proof. By definition

$$\begin{aligned}
\langle \mathbf{c}, (1, \mathbf{s}) \rangle &= \left\langle \mathbf{P}^T \cdot \mathbf{r} + \left\lfloor \frac{q}{2} \right\rfloor \cdot \mathbf{m}, (1, \mathbf{s}) \right\rangle \pmod{q} \\
&= \left\lfloor \frac{q}{2} \right\rfloor \cdot m + \mathbf{r}^T \mathbf{P} \cdot (1, \mathbf{s}) \pmod{q} \\
&= \left\lfloor \frac{q}{2} \right\rfloor \cdot m + \mathbf{r}^T \mathbf{b} - \mathbf{r}^T \mathbf{A} \mathbf{s} \pmod{q} \\
&= \left\lfloor \frac{q}{2} \right\rfloor \cdot m + \langle \mathbf{r}, \mathbf{e} \rangle \pmod{q} .
\end{aligned}$$

The lemma follows since $|\langle \mathbf{r}, \mathbf{e} \rangle| \leq N \cdot B$. \square

We proceed to state the correctness of decryption for low-noise ciphertexts. The proof easily follows by assignment into the definition of Regev.Dec and is omitted.

Lemma 3.2 (decryption noise). *Let $\mathbf{s} \in \mathbb{Z}^n$ be some vector, and let $\mathbf{c} \in \mathbb{Z}_q^{n+1}$ be such that*

$$\langle \mathbf{c}, (1, \mathbf{s}) \rangle = \left\lfloor \frac{q}{2} \right\rfloor \cdot m + e \pmod{q} ,$$

with $m \in \{0, 1\}$ and $|e| < \lfloor q/2 \rfloor / 2$. Then

$$\text{Regev.Dec}_s(\mathbf{c}) = m .$$

Security. The following lemma states the security of Regev . The proof is standard (see e.g. [Reg05]) and is omitted.

Lemma 3.3. *Let n, q, χ be some parameters such that $\text{DLWE}_{n, q, \chi}$ holds. Then for any $m \in \{0, 1\}$, if $\mathbf{s} \leftarrow \text{Regev.SecretKeygen}(1^n)$, $\mathbf{P} \leftarrow \text{Regev.PublicKeygen}(\mathbf{s})$, $\mathbf{c} \leftarrow \text{Regev.Enc}_{\mathbf{P}}(m)$, it holds that the joint distribution (\mathbf{P}, \mathbf{c}) is computationally indistinguishable from uniform over $\mathbb{Z}_q^{N \times (n+1)} \times \mathbb{Z}_q^{n+1}$.*

3.2 Vector Decomposition and Key Switching

We show how to decompose vectors in a way that preserves inner product and how to generate and use key switching parameters. Our notation is generally adopted from [BGV12].

Vector Decomposition. We often break vectors into their bit representations as defined below:

- $\text{BitDecomp}_q(\mathbf{x})$: For $\mathbf{x} \in \mathbb{Z}^n$, let $\mathbf{w}_i \in \{0, 1\}^n$ be such that $\mathbf{x} = \sum_{i=0}^{\lceil \log q \rceil - 1} 2^i \cdot \mathbf{w}_i \pmod{q}$. Output the vector

$$(\mathbf{w}_0, \dots, \mathbf{w}_{\lceil \log q \rceil - 1}) \in \{0, 1\}^{n \cdot \lceil \log q \rceil} .$$

- $\text{PowersOfTwo}_q(\mathbf{y})$: For $\mathbf{y} \in \mathbb{Z}^n$, output

$$\left[(\mathbf{y}, 2 \cdot \mathbf{y}, \dots, 2^{\lceil \log q \rceil - 1} \cdot \mathbf{y}) \right]_q \in \mathbb{Z}_q^{n \cdot \lceil \log q \rceil} .$$

We will usually omit the subscript q when it is clear from the context.

Claim 3.4. *For all $q \in \mathbb{Z}$ and $\mathbf{x}, \mathbf{y} \in \mathbb{Z}^n$, it holds that*

$$\langle \mathbf{x}, \mathbf{y} \rangle = \langle \text{BitDecomp}_q(\mathbf{x}), \text{PowersOfTwo}_q(\mathbf{y}) \rangle \pmod{q} .$$

Key Switching. In the functions below, q is an integer and χ is a distribution over \mathbb{Z} :

- $\text{SwitchKeyGen}_{q,\chi}(\mathbf{s}, \mathbf{t})$: For a “source” key $\mathbf{s} \in \mathbb{Z}^{n_s}$ and “target” key $\mathbf{t} \in \mathbb{Z}^{n_t}$, we define a set of parameters that allow to switch ciphertexts under \mathbf{s} into ciphertexts under $(1, \mathbf{t})$.

Let $\hat{n}_s \triangleq n_s \cdot \lceil \log q \rceil$ be the dimension of $\text{PowersOfTwo}_q(\mathbf{s})$. Sample a uniform matrix $\mathbf{A}_{\mathbf{s}:\mathbf{t}} \xleftarrow{\$} \mathbb{Z}_q^{\hat{n}_s \times n_t}$ and a noise vector $\mathbf{e} \xleftarrow{\$} \chi^{\hat{n}_s}$. The function’s output is a matrix

$$\mathbf{P}_{\mathbf{s}:\mathbf{t}} = [\mathbf{b}_{\mathbf{s}:\mathbf{t}} \parallel -\mathbf{A}_{\mathbf{s}:\mathbf{t}}] \in \mathbb{Z}_q^{\hat{n}_s \times (n_t+1)},$$

where

$$\mathbf{b}_{\mathbf{s}:\mathbf{t}} := \left[\mathbf{A}_{\mathbf{s}:\mathbf{t}} \cdot \mathbf{t} + \mathbf{e}_{\mathbf{s}:\mathbf{t}} + \text{PowersOfTwo}_q(\mathbf{s}) \right]_q \in \mathbb{Z}_q^{\hat{n}_s}.$$

This is similar, although not identical, to encrypting $\text{PowersOfTwo}_q(\mathbf{s})$ (the difference is that $\text{PowersOfTwo}_q(\mathbf{s})$ contains non-binary values).

- $\text{SwitchKey}_q(\mathbf{P}_{\mathbf{s}:\mathbf{t}}, \mathbf{c}_s)$: To switch a ciphertext from a secret key \mathbf{s} to $(1, \mathbf{t})$, output

$$\mathbf{c}_t := [\mathbf{P}_{\mathbf{s}:\mathbf{t}}^T \cdot \text{BitDecomp}_q(\mathbf{c}_s)]_q.$$

Again, we usually omit the subscripts when they are clear from the context. Correctness and security are stated below, the proofs are by definition.

Lemma 3.5 (correctness). *Let $\mathbf{s} \in \mathbb{Z}^{n_s}$, $\mathbf{t} \in \mathbb{Z}^{n_t}$ and $\mathbf{c}_s \in \mathbb{Z}_q^{n_s}$ be any vectors. Let $\mathbf{P}_{\mathbf{s}:\mathbf{t}} \leftarrow \text{SwitchKeyGen}(\mathbf{s}, \mathbf{t})$ and set $\mathbf{c}_t \leftarrow \text{SwitchKey}(\mathbf{P}_{\mathbf{s}:\mathbf{t}}, \mathbf{c}_s)$. Then*

$$\langle \mathbf{c}_s, \mathbf{s} \rangle = \langle \mathbf{c}_t, (1, \mathbf{t}) \rangle - \langle \text{BitDecomp}_q(\mathbf{c}_s), \mathbf{e}_{\mathbf{s}:\mathbf{t}} \rangle \pmod{q}.$$

Lemma 3.6 (security). *Let $\mathbf{s} \in \mathbb{Z}^{n_s}$ be any vector. If we generate $\mathbf{t} \leftarrow \text{Regev.SecretKeygen}(1^n)$ and $\mathbf{P} \leftarrow \text{SwitchKeyGen}_{q,\chi}(\mathbf{s}, \mathbf{t})$, then \mathbf{P} is computationally indistinguishable from uniform over $\mathbb{Z}_q^{\hat{n}_s \times (n_t+1)}$, assuming $\text{DLWE}_{n,q,\chi}$.*

4 A Scale Invariant Homomorphic Encryption Scheme

We present our scale invariant L -homomorphic scheme as outlined in Section 1.2. Homomorphic properties are discussed in Section 4.1, implications and optimizations are discussed in Section 4.2.

Let $q = q(n)$ be an integer function, let $L = L(n)$ be a polynomial and let $\chi = \chi(n)$ be a distribution ensemble over \mathbb{Z} . The scheme SI-HE is defined as follows:

- $\text{SI-HE.Keygen}(1^L, 1^n)$: Sample $L+1$ vectors $\mathbf{s}_0, \dots, \mathbf{s}_L \leftarrow \text{Regev.SecretKeygen}(1^n)$, and compute a Regev public key for the first one: $\mathbf{P}_0 \leftarrow \text{Regev.PublicKeygen}(\mathbf{s}_0)$. For all $i \in [L]$, define

$$\tilde{\mathbf{s}}_{i-1} := \text{BitDecomp}((1, \mathbf{s}_{i-1})) \otimes \text{BitDecomp}((1, \mathbf{s}_{i-1})) \in \{0, 1\}^{((n+1)\lceil \log q \rceil)^2}.$$

and compute

$$\mathbf{P}_{(i-1):i} \leftarrow \text{SwitchKeyGen}(\tilde{\mathbf{s}}_{i-1}, \mathbf{s}_i).$$

Output $pk = \mathbf{P}_0$, $evk = \{\mathbf{P}_{(i-1):i}\}_{i \in [L]}$ and $sk = \mathbf{s}_L$.

- $\text{SI-HE.Enc}_{pk}(m)$: Identical to Regev's, output $\mathbf{c} \leftarrow \text{Regev.Enc}_{pk}(m)$.
- $\text{SI-HE.Eval}_{evk}(\cdot)$: As usual, we describe homomorphic addition and multiplication over $\text{GF}(2)$, which allows to evaluate depth L arithmetic circuits in a gate-by-gate manner. The convention for a gate at level i of the circuit is that the operand ciphertexts are decryptable using \mathbf{s}_{i-1} , and the output of the homomorphic operation is decryptable using \mathbf{s}_i .

Since evk contains key switching parameters from $\tilde{\mathbf{s}}_{i-1}$ to \mathbf{s}_i , homomorphic addition and multiplication both first produce an intermediate output $\tilde{\mathbf{c}}$ that corresponds to $\tilde{\mathbf{s}}_{i-1}$, and then use key switching to obtain the final output.¹⁰

- $\text{SI-HE.Add}_{evk}(\mathbf{c}_1, \mathbf{c}_2)$: Assume w.l.o.g that both input ciphertexts are encrypted under the same secret key \mathbf{s}_{i-1} . First compute

$$\tilde{\mathbf{c}}_{\text{add}} := \text{PowersOfTwo}(\mathbf{c}_1 + \mathbf{c}_2) \otimes \text{PowersOfTwo}((1, 0, \dots, 0)) ,$$

then output

$$\mathbf{c}_{\text{add}} \leftarrow \text{SwitchKey}(\mathbf{P}_{(i-1):i}, \tilde{\mathbf{c}}_{\text{add}}) \in \mathbb{Z}_q^{n+1} .$$

Let us explain what we did: We first added the ciphertext vectors (as expected) to obtain $\mathbf{c}_1 + \mathbf{c}_2$. This already implements the homomorphic addition, but provides an output that corresponds to \mathbf{s}_{i-1} and not \mathbf{s}_i as required. We thus generate $\tilde{\mathbf{c}}_{\text{add}}$ by tensoring with a “trivial” ciphertext. The result corresponds to $\tilde{\mathbf{s}}_{i-1}$, and allows to finally use key switching to obtain an output corresponding to \mathbf{s}_i . We use powers-of-two representation in order to control the norm of the secret key (as we explain in Section 1.2).

- $\text{SI-HE.Mult}_{evk}(\mathbf{c}_1, \mathbf{c}_2)$: Assume w.l.o.g that both input ciphertexts are encrypted under the same secret key \mathbf{s}_{i-1} . First compute

$$\tilde{\mathbf{c}}_{\text{mult}} := \left\lfloor \frac{2}{q} \cdot \left(\text{PowersOfTwo}(\mathbf{c}_1) \otimes \text{PowersOfTwo}(\mathbf{c}_2) \right) \right\rfloor ,$$

then output

$$\mathbf{c}_{\text{mult}} \leftarrow \text{SwitchKey}(\mathbf{P}_{(i-1):i}, \tilde{\mathbf{c}}_{\text{mult}}) \in \mathbb{Z}_q^{n+1} .$$

As we explain in Section 1.2, The tensored ciphertext $\tilde{\mathbf{c}}_{\text{mult}}$ mimics tensoring in the “invariant perspective”, which produces an encryption of the product of the plaintexts under the tensored secret key $\tilde{\mathbf{s}}_{i-1}$. We then switch keys to obtain an output corresponding to \mathbf{s}_i .

- Decryption $\text{SI-HE.Dec}_{sk}(\mathbf{c})$: Assume w.l.o.g that \mathbf{c} is a ciphertext that corresponds to $\mathbf{s}_L (=sk)$. Then decryption is again identical to Regev's, output

$$m \leftarrow \text{Regev.Dec}_{sk}(\mathbf{c}) .$$

¹⁰The final key switching replaces the more complicated “refresh” operation of [BGV12].

Security. The security of the scheme follows in a straightforward way, very similarly to the proof of [BV11b, Theorem 4.1] as we sketch below.

Lemma 4.1. *Let n, q, χ be some parameters such that $\text{DLWE}_{n,q,\chi}$ holds, and let $L = L(n)$ be polynomially bounded. Then for any $m \in \{0, 1\}$, if $(pk, evk, sk) \leftarrow \text{SI-HE.Keygen}(1^L, 1^n)$, $\mathbf{c} \leftarrow \text{SI-HE.Enc}_{pk}(m)$, it holds that the joint distribution (pk, evk, \mathbf{c}) is computationally indistinguishable from uniform.*

Proof sketch. We consider the distribution $(pk, evk, \mathbf{c}) = (\mathbf{P}_0, \mathbf{P}_{0:1}, \dots, \mathbf{P}_{L-1:L}, \mathbf{c})$ and apply a hybrid argument.

First, we argue that $\mathbf{P}_{L-1:L}$ is indistinguishable from uniform, based on Lemma 3.6 (note that \mathbf{s}_L is only used to generate $\mathbf{P}_{L-1:L}$). We then proceed to replace all $\mathbf{P}_{i-1:i}$ with uniform in descending order, based on the same argument. Finally, we are left with $(\mathbf{P}_0, \mathbf{c})$ (and a multitude of uniform elements), which are exactly a public key and ciphertext of Regev's scheme. We invoke Lemma 3.3 to argue that $(\mathbf{P}_0, \mathbf{c})$ are indistinguishable from uniform, which completes the proof of our lemma.

We remark that generally one has to be careful when using a super-constant number of hybrids, but in our case, as in [BV11b], this causes no problem. \square

4.1 Homomorphic Properties of SI-HE

The following theorem summarizes the homomorphic properties of our scheme.

Theorem 4.2. *The scheme SI-HE with parameters $n, q, |\chi| \leq B, L$ for which*

$$q/B \geq (O(n \log q))^{L+O(1)} ,$$

is L -homomorphic.

The theorem is proven using the following lemma, which bounds the growth of the noise in gate evaluation.

Lemma 4.3. *Let $q, n, |\chi| \leq B, L$ be parameters for SI-HE, and let $(pk, evk, sk) \leftarrow \text{SI-HE.Keygen}(1^L, 1^n)$. Let $\mathbf{c}_1, \mathbf{c}_2$ be such that*

$$\begin{aligned} \langle \mathbf{c}_1, (1, \mathbf{s}_{i-1}) \rangle &= \left\lfloor \frac{q}{2} \right\rfloor \cdot m_1 + e_1 \pmod{q} \\ \langle \mathbf{c}_2, (1, \mathbf{s}_{i-1}) \rangle &= \left\lfloor \frac{q}{2} \right\rfloor \cdot m_1 + e_2 \pmod{q} , \end{aligned} \tag{1}$$

with $|e_1|, |e_2| \leq E < \lfloor q/2 \rfloor / 2$. Define $\mathbf{c}_{\text{add}} \leftarrow \text{SI-HE.Add}_{evk}(\mathbf{c}_1, \mathbf{c}_2)$, $\mathbf{c}_{\text{mult}} \leftarrow \text{SI-HE.Mult}_{evk}(\mathbf{c}_1, \mathbf{c}_2)$. Then

$$\begin{aligned} \langle \mathbf{c}_{\text{add}}, (1, \mathbf{s}_i) \rangle &= \left\lfloor \frac{q}{2} \right\rfloor \cdot ([m_1 + m_2]_2) + e_{\text{add}} \pmod{q} \\ \langle \mathbf{c}_{\text{mult}}, (1, \mathbf{s}_i) \rangle &= \left\lfloor \frac{q}{2} \right\rfloor \cdot m_1 m_2 + e_{\text{mult}} \pmod{q} , \end{aligned}$$

where

$$|e_{\text{add}}|, |e_{\text{mult}}| \leq O(n \log q) \cdot \max \{ E, (n \log^2 q) \cdot B \} .$$

We remark that, as usual, homomorphic addition increases noise much more moderately than multiplication, but the coarse bound we show in the lemma is sufficient for our purposes.

Next we show how to use Lemma 4.3 to prove Theorem 4.2. The proof of Lemma 4.3 itself is deferred to Section 4.3.

Proof of Theorem 4.2. Consider the evaluation of a depth L circuit. Let E_i be a bound on the noise in the ciphertext after the evaluation of the i^{th} level of gates.

By Lemma 3.1, $E_0 = N \cdot B = O(n \log q) \cdot B$. Lemma 4.3 guarantees that starting from the point where $E \geq (n \log^2 q) \cdot B$, it will hold that $E_{i+1} = O(n \log q) \cdot E_i$. We get that $E_L = (O(n \log q))^{L+O(1)} \cdot B$.

By Lemma 3.2, decryption will succeed if $E_L < \lfloor q/2 \rfloor / 2$ and the theorem follows. \square

4.2 Implications and Optimizations

Fully Homomorphic Encryption using Bootstrapping. Fully homomorphic encryption follows using the bootstrapping theorem (Theorem 2.2). In order to use bootstrapping, we need to bound the depth of the decryption circuit. The following lemma has been proven in a number of previous works (e.g. [BV11b, Lemma 4.5]):

Lemma 4.4. *For all \mathbf{c} , the function $f_{\mathbf{c}}(\mathbf{s}) = \text{SI-HE.Dec}_{\mathbf{s}}(\mathbf{c})$ can be implemented by a circuit of depth $O(\log n + \log \log q)$.*

An immediate corollary follows from Theorem 2.2, Theorem 4.2 and Lemma 4.4:

Corollary 4.5. *Let n, q, χ, B be such that $|\chi| \leq B$ and $q/B \geq (n \log q)^{O(\log n + \log \log q)}$. Then there exists a (leveled) fully homomorphic encryption scheme based on the $\text{DLWE}_{n,q,\chi}$ assumption.*

Furthermore, if SI-HE is weak circular secure, then the same assumption implies full (non leveled) homomorphism.

Finally, we can classically reduce security from GapSVP using Corollary 2.1, by choosing $q = \tilde{O}(2^{n/2})$ and $B = q/(n \log q)^{O(\log n + \log \log q)} = q/n^{O(\log n)}$:

Corollary 4.6. *There exists a (leveled) fully-homomorphic encryption scheme based on the classical worst case hardness of the $\text{GapSVP}_{n^{O(\log n)}}$ problem.*

(Leveled) Fully Homomorphic Encryption without Bootstrapping. Following [BGV12], our scheme implies a leveled fully homomorphic encryption without bootstrapping. Plugging our scheme into the [BGV12] framework, we obtain a (leveled) fully homomorphic encryption without bootstrapping, based on the classical worst case hardness of $\text{GapSVP}_{2^{n^\epsilon}}$, for any $\epsilon > 0$.

Optimizations. So far, we chose to present our scheme in the cleanest possible way. However, there are a few techniques that can somewhat improve performance. While the asymptotic advantage of some of these methods is not great, a real life implementation can benefit from them.

1. Our tensored secret key $\tilde{\mathbf{s}}_{i-1}$ is obtained by tensoring a vector with itself. Such a vector can be represented by only $\binom{n_s}{2}$ (as opposed to our n_s^2), saving a factor of (almost) 2 in the representation length.
2. When $B \ll q$, some improvement can be achieved by using LWE in Hermite normal form. It is known (see e.g. [ACPS09]) that the hardness of LWE remains essentially the same if we sample $\mathbf{s} \xleftarrow{\$} \chi^n$ (instead of uniformly in \mathbb{Z}_q^n). Sampling our keys this way, we only need $O(n \log B)$ bits to represent $\text{BitDecomp}(\mathbf{s})$, and its norm goes down accordingly.

We can therefore reduce the size of the evaluation key (which depends quadratically on the bit length of the secret key), and more importantly, we can prove a tighter version of Lemma 4.3. When using Hermite normal form, the noise grows from E to $O(n \log B) \cdot \max\{E, (n \log B \log q) \cdot B\}$. Therefore, L -homomorphism is achieved whenever

$$q/B \geq (O(n \log B))^{L+O(1)} \cdot \log q .$$

3. The least significant bits of the ciphertext can sometimes be truncated without much harm, which can lead to significant saving in ciphertext and key length, especially when B/q is large: Let \mathbf{c} be an integer ciphertext vector and define $\mathbf{c}' = \lfloor 2^{-i} \cdot \mathbf{c} \rfloor$. Then \mathbf{c}' , which can be represented with $n \cdot i$ fewer bits than \mathbf{c} , implies a good approximation for \mathbf{c} since

$$|\langle \mathbf{c}, \mathbf{s} \rangle - \langle 2^i \cdot \mathbf{c}', \mathbf{s} \rangle| \leq 2^{i-1} \|\mathbf{s}\|_1 .$$

This means that $2^i \cdot \mathbf{c}'$ can be used instead of \mathbf{c} , at the cost of an additive increase in the noise magnitude.

Consider a case where $q, B \gg q/B$ (which occurs when we artificially increase q in order for the classical reduction to work). Recall that $\|\mathbf{s}\|_1 \approx n \log q$ and consider truncating with $i \approx \log(B/(n \log q))$. Then the additional noise incurred by using \mathbf{c}' instead of \mathbf{c} is only an insignificant $\approx B$. The number of bits required to represent each element in \mathbf{c}' however now becomes $\log q - i \approx \log(q/B) + \log(n \log q)$. In conclusion, we hardly lose anything in ciphertext length compared to the case of working with smaller q, B to begin with (with similar q/B ratio). The ciphertext length can, therefore, be made invariant to the absolute values of q, B , and depend only on their ratio. This of course applies also to the vectors in *evk*.

4.3 Proof of Lemma 4.3

We start with the analysis for addition, which is simpler and will also serve as good warm-up towards the analysis for multiplication.

Analysis for Addition. By Lemma 3.5, it holds that

$$\langle \mathbf{c}_{\text{add}}, (1, \mathbf{s}_i) \rangle = \langle \tilde{\mathbf{c}}_{\text{add}}, \tilde{\mathbf{s}}_i \rangle + \underbrace{\langle \text{BitDecomp}(\tilde{\mathbf{c}}), \mathbf{e}_{i-1:i} \rangle}_{\triangleq \delta_1} \pmod{q} .$$

where $\mathbf{e}_{i-1:i} \sim \chi^{(n+1)^2 \cdot (\lceil \log q \rceil)^3}$. That is, δ_1 is the noise inflicted by the key switching process.

We bound $|\delta_1|$ using the bound on χ :

$$|\delta_1| = |\langle \text{BitDecomp}(\tilde{\mathbf{c}}_{\text{add}}), \mathbf{e}_{i-1:i} \rangle| \leq (n+1)^2 \cdot (\lceil \log q \rceil)^3 \cdot B = O(n^2 \log^3 q) \cdot B .$$

Next, we expand the term $\langle \tilde{\mathbf{c}}_{\text{add}}, \tilde{\mathbf{s}}_i \rangle$, by breaking an inner product of tensors into a product of inner products (one of which is trivially equal to 1):

$$\begin{aligned} \langle \tilde{\mathbf{c}}_{\text{add}}, \tilde{\mathbf{s}}_i \rangle &= \left\langle \text{PowersOfTwo}(\mathbf{c}_1 + \mathbf{c}_2) \otimes \text{PowersOfTwo}((1, 0, \dots, 0)), \right. \\ &\quad \left. \text{BitDecomp}((1, \mathbf{s}_{i-1})) \otimes \text{BitDecomp}((1, \mathbf{s}_{i-1})) \right\rangle \\ &= \langle \text{PowersOfTwo}(\mathbf{c}_1 + \mathbf{c}_2), \text{BitDecomp}((1, \mathbf{s}_{i-1})) \rangle \cdot 1 \\ &= \langle (\mathbf{c}_1 + \mathbf{c}_2), (1, \mathbf{s}_{i-1}) \rangle \pmod{q} \\ &= \langle \mathbf{c}_1, (1, \mathbf{s}_{i-1}) \rangle + \langle \mathbf{c}_2, (1, \mathbf{s}_{i-1}) \rangle \pmod{q} . \end{aligned}$$

We can now plug in what we know about $\mathbf{c}_1, \mathbf{c}_2$ from Eq. (1) in the lemma statement:

$$\begin{aligned}\langle \tilde{\mathbf{c}}_{\text{add}}, \tilde{\mathbf{s}}_i \rangle &= \left\lfloor \frac{q}{2} \right\rfloor \cdot m_1 + e_1 + \left\lfloor \frac{q}{2} \right\rfloor \cdot m_2 + e_2 \pmod{q} \\ &= \left\lfloor \frac{q}{2} \right\rfloor \cdot [m_1 + m_2]_2 \underbrace{- \tilde{m} + e_1 + e_2}_{\triangleq \delta_2} \pmod{q}\end{aligned}$$

where $\tilde{m} \in \{0, 1\}$ is defined as:

$$\tilde{m} \triangleq \begin{cases} 0, & \text{if } q \text{ is even,} \\ \frac{1}{2} \cdot (m_1 + m_2 - [m_1 + m_2]_2), & \text{if } q \text{ is odd,} \end{cases}$$

and $|\delta_2| \leq 1 + 2E$.

Putting it all together,

$$\langle \mathbf{c}_{\text{add}}, (1, \mathbf{s}_i) \rangle = \left\lfloor \frac{q}{2} \right\rfloor \cdot [m_1 + m_2]_2 + \underbrace{\delta_1 + \delta_2}_{=e_{\text{add}}} \pmod{q}.$$

Where the bound on e_{add} is

$$|e_{\text{add}}| = |\delta_1 + \delta_2| \leq O(n^2 \log^3 q) \cdot B + O(1) \cdot E \leq O(n \log q) \cdot \max \{E, (n \log^2 q) \cdot B\}.$$

This finishes the argument for addition.

Analysis for Multiplication. The analysis for multiplication starts very similarly to addition:

$$\langle \mathbf{c}_{\text{mult}}, (1, \mathbf{s}_i) \rangle = \langle \tilde{\mathbf{c}}_{\text{mult}}, \tilde{\mathbf{s}}_i \rangle + \underbrace{\langle \text{BitDecomp}(\tilde{\mathbf{c}}_{\text{mult}}), \mathbf{e}_{i-1:i} \rangle}_{\triangleq \delta_1} \pmod{q},$$

and as before

$$|\delta_1| = O(n^2 \log^3 q) \cdot B.$$

Let us now focus on $\langle \tilde{\mathbf{c}}_{\text{mult}}, \tilde{\mathbf{s}}_i \rangle$. We want to use the properties of tensoring to break the inner product into two smaller inner products, as we did before. This time, however, $\tilde{\mathbf{c}}_{\text{mult}}$ is a *rounded* tensor:

$$\langle \tilde{\mathbf{c}}, \tilde{\mathbf{s}}_i \rangle = \left\langle \left\lfloor \frac{2}{q} \cdot (\text{PowersOfTwo}(\mathbf{c}_1) \otimes \text{PowersOfTwo}(\mathbf{c}_2)) \right\rfloor, \tilde{\mathbf{s}}_{i-1} \right\rangle \pmod{q}.$$

We start by showing that the rounding does not add much noise. Intuitively this is because $\tilde{\mathbf{s}}_{i-1}$ is a binary vector and thus has low norm. We define

$$\begin{aligned}\delta_2 \triangleq & \left\langle \left\lfloor \frac{2}{q} \cdot (\text{PowersOfTwo}(\mathbf{c}_1) \otimes \text{PowersOfTwo}(\mathbf{c}_2)) \right\rfloor, \tilde{\mathbf{s}}_{i-1} \right\rangle \\ & - \left\langle \frac{2}{q} \cdot (\text{PowersOfTwo}(\mathbf{c}_1) \otimes \text{PowersOfTwo}(\mathbf{c}_2)), \tilde{\mathbf{s}}_{i-1} \right\rangle,\end{aligned}$$

and for convenience we also define

$$\begin{aligned}\mathbf{c}' \triangleq & \left\lfloor \frac{2}{q} \cdot (\text{PowersOfTwo}(\mathbf{c}_1) \otimes \text{PowersOfTwo}(\mathbf{c}_2)) \right\rfloor \\ & - \frac{2}{q} \cdot (\text{PowersOfTwo}(\mathbf{c}_1) \otimes \text{PowersOfTwo}(\mathbf{c}_2)).\end{aligned}$$

By definition, $\delta_2 = \langle \mathbf{c}', \tilde{\mathbf{s}}_{i-1} \rangle$. Now, since $\|\mathbf{c}'\|_\infty \leq 1/2$ and $\|\tilde{\mathbf{s}}_{i-1}\|_1 \leq ((n+1) \lceil \log q \rceil)^2 = O(n^2 \log^2 q)$, it follows that

$$|\delta_2| \leq \|\mathbf{c}'\|_\infty \cdot \|\tilde{\mathbf{s}}_{i-1}\|_1 = O(n^2 \log^2 q) .$$

We can now break the inner product using the properties of tensoring:

$$\begin{aligned} \langle \tilde{\mathbf{c}}_{\text{mult}}, \tilde{\mathbf{s}}_{i-1} \rangle - \delta_2 &= \frac{2}{q} \cdot \langle \text{PowersOfTwo}(\mathbf{c}_1), \text{BitDecomp}((1, \mathbf{s}_{i-1})) \rangle \\ &\quad \cdot \langle \text{PowersOfTwo}(\mathbf{c}_2), \text{BitDecomp}((1, \mathbf{s}_{i-1})) \rangle . \end{aligned} \quad (2)$$

Note that we keep $\mathbf{c}_1, \mathbf{c}_2$ in powers-of-two form. This is deliberate and will be useful later (essentially because we want our ciphertext to relate to low-norm secrets).

Going back to Eq. (1) from the lemma statement, it follows (using Claim 3.4) that

$$\begin{aligned} \langle \text{PowersOfTwo}(\mathbf{c}_1), \text{BitDecomp}(1, \mathbf{s}_{i-1}) \rangle &= \left\lfloor \frac{q}{2} \right\rfloor \cdot m_1 + e_1 \pmod{q} \\ \langle \text{PowersOfTwo}(\mathbf{c}_2), \text{BitDecomp}(1, \mathbf{s}_{i-1}) \rangle &= \left\lfloor \frac{q}{2} \right\rfloor \cdot m_2 + e_2 \pmod{q} . \end{aligned}$$

Let $I_1, I_2 \in \mathbb{Z}$ be such that

$$\begin{aligned} \langle \text{PowersOfTwo}(\mathbf{c}_1), \text{BitDecomp}(1, \mathbf{s}_{i-1}) \rangle &= \left\lfloor \frac{q}{2} \right\rfloor \cdot m_1 + e_1 + q \cdot I_1 \\ \langle \text{PowersOfTwo}(\mathbf{c}_2), \text{BitDecomp}(1, \mathbf{s}_{i-1}) \rangle &= \left\lfloor \frac{q}{2} \right\rfloor \cdot m_2 + e_2 + q \cdot I_2 . \end{aligned} \quad (3)$$

Let us bound the absolute value of I_1 (obviously the same bound also holds for I_2):

$$\begin{aligned} |I_1| &= \frac{|\langle \text{PowersOfTwo}(\mathbf{c}_1), \text{BitDecomp}(1, \mathbf{s}_{i-1}) \rangle - \left\lfloor \frac{q}{2} \right\rfloor \cdot m_1 - e_1|}{q} \\ &\leq \frac{|\langle \text{PowersOfTwo}(\mathbf{c}_1), \text{BitDecomp}(1, \mathbf{s}_{i-1}) \rangle|}{q} + 1 \\ &\leq \frac{\|\text{PowersOfTwo}(\mathbf{c}_1)\|_\infty}{q} \cdot \|\text{BitDecomp}(1, \mathbf{s}_{i-1})\|_1 + 1 \\ &\leq \frac{1}{2} \cdot \|\text{BitDecomp}(1, \mathbf{s}_{i-1})\|_1 + 1 \\ &\leq \frac{1}{2} \cdot (n+1) \lceil \log q \rceil + 1 \\ &= O(n \log q) . \end{aligned} \quad (4)$$

Plugging Eq. (3) into Eq. (2), we get

$$\begin{aligned} \langle \tilde{\mathbf{c}}_{\text{mult}}, \tilde{\mathbf{s}}_{i-1} \rangle - \delta_2 &= \frac{2}{q} \cdot \left(\left\lfloor \frac{q}{2} \right\rfloor \cdot m_1 + e_1 + q \cdot I_1 \right) \cdot \left(\left\lfloor \frac{q}{2} \right\rfloor \cdot m_2 + e_2 + q \cdot I_2 \right) \\ &= \left\lfloor \frac{q}{2} \right\rfloor \cdot m_1 \cdot m_2 + \delta_3 + q \cdot (m_1 I_2 + m_2 I_1 + 2I_1 I_2) , \end{aligned}$$

where δ_3 is defined as

$$\delta_3 \triangleq \begin{cases} 2e_2 \cdot I_1 + 2e_1 \cdot I_2 + (e_1 m_2 + e_2 m_1) + \frac{2e_1 \cdot e_2}{q} , & \text{if } q \text{ is even,} \\ (2e_2 - m_2) \cdot I_1 + (2e_1 - m_1) \cdot I_2 + \frac{q-1}{q} \cdot (e_1 m_2 + e_2 m_1) - \frac{m_1 \cdot m_2}{2q} + \frac{2e_1 \cdot e_2}{q} , & \text{if } q \text{ is odd.} \end{cases}$$

In particular (recall that $E < \lfloor q/2 \rfloor / 2 \leq q/4$):

$$|\delta_3| \leq 2(2E + 1)O(n \log q) + 2E + \frac{1 + 2E^2}{q} = O(n \log q) \cdot E .$$

Putting everything together, we get that

$$\langle \mathbf{c}_{\text{mult}}, (1, \mathbf{s}_i) \rangle = \left\lfloor \frac{q}{2} \right\rfloor \cdot m_1 m_2 + \underbrace{\delta_1 + \delta_2 + \delta_3}_{=e_{\text{mult}}} \pmod{q} ,$$

where

$$|e_{\text{mult}}| = |\delta_1 + \delta_2 + \delta_3| \leq O(n \log q) \cdot E + O(n^2 \log^3 q) \cdot B ,$$

and the lemma follows. \square

Acknowledgments

We thank Vinod Vaikuntanathan for fruitful discussions and advice, and Dan Boneh for his comments on an earlier version of this manuscript. We thank the reviewers of CRYPTO 2012 for their constructive comments. In addition, we thank various readers for pointing out typos in earlier versions of this manuscript.

References

- [ACPS09] Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In Shai Halevi, editor, *CRYPTO*, volume 5677 of *Lecture Notes in Computer Science*, pages 595–618. Springer, 2009.
- [AD97] Miklós Ajtai and Cynthia Dwork. A public-key cryptosystem with worst-case/average-case equivalence. In Frank Thomson Leighton and Peter W. Shor, editors, *STOC*, pages 284–293. ACM, 1997.
- [BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ITCS*, 2012. See also <http://eprint.iacr.org/2011/277>.
- [BV11a] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-LWE and security for key dependent messages. In *CRYPTO*, volume 6841, page 501, 2011.
- [BV11b] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In Ostrovsky [Ost11], pages 97–106. References are to full version: <http://eprint.iacr.org/2011/344>.
- [CMNT11] Jean-Sébastien Coron, Avradip Mandal, David Naccache, and Mehdi Tibouchi. Fully homomorphic encryption over the integers with shorter public keys. In Rogaway [Rog11], pages 487–504.

- [DGHV10] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *EUROCRYPT*, pages 24–43, 2010. Full Version in <http://eprint.iacr.org/2009/616.pdf>.
- [Gen09a] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. <http://crypto.stanford.edu/craig>.
- [Gen09b] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009.
- [GGH97] Oded Goldreich, Shafi Goldwasser, and Shai Halevi. Eliminating decryption errors in the ajtai-dwork cryptosystem. In Burton S. Kaliski Jr., editor, *CRYPTO*, volume 1294 of *Lecture Notes in Computer Science*, pages 105–111. Springer, 1997.
- [GH11] Craig Gentry and Shai Halevi. Fully homomorphic encryption without squashing using depth-3 arithmetic circuits. In Ostrovsky [Ost11], pages 107–109.
- [GHS11a] Craig Gentry, Shai Halevi, and Nigel P. Smart. Better bootstrapping in fully homomorphic encryption. *IACR Cryptology ePrint Archive*, 2011:680, 2011.
- [GHS11b] Craig Gentry, Shai Halevi, and Nigel P. Smart. Fully homomorphic encryption with polylog overhead. *IACR Cryptology ePrint Archive*, 2011:566, 2011.
- [GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In Cynthia Dwork, editor, *STOC*, pages 197–206. ACM, 2008.
- [LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *EUROCRYPT*, pages 1–23, 2010. Draft of full version was provided by the authors.
- [MM11] Daniele Micciancio and Petros Mol. Pseudorandom knapsacks and the sample complexity of lwe search-to-decision reductions. In Rogaway [Rog11], pages 465–484.
- [MP11] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. *IACR Cryptology ePrint Archive*, 2011:501, 2011. To appear in Eurocrypt 2012.
- [MV10] Daniele Micciancio and Panagiotis Voulgaris. A deterministic single exponential time algorithm for most lattice problems based on voronoi cell computations. In Leonard J. Schulman, editor, *STOC*, pages 351–358. ACM, 2010.
- [Ost11] Rafail Ostrovsky, editor. *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*. IEEE, 2011.
- [Pei09] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In *STOC*, pages 333–342, 2009.
- [RAD78] R. Rivest, L. Adleman, and M. Dertouzos. On data banks and privacy homomorphisms. In *Foundations of Secure Computation*, pages 169–177. Academic Press, 1978.

- [Reg03] Oded Regev. New lattice based cryptographic constructions. In Lawrence L. Larmore and Michel X. Goemans, editors, *STOC*, pages 407–416. ACM, 2003. Full version in [Reg04].
- [Reg04] Oded Regev. New lattice-based cryptographic constructions. *J. ACM*, 51(6):899–942, 2004.
- [Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *STOC*, pages 84–93. ACM, 2005. Full version in [Reg09].
- [Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6), 2009.
- [Rog11] Phillip Rogaway, editor. *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, volume 6841 of *Lecture Notes in Computer Science*. Springer, 2011.
- [Sch87] Claus-Peter Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theor. Comput. Sci.*, 53:201–224, 1987.
- [SV10] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In Phong Q. Nguyen and David Pointcheval, editors, *Public Key Cryptography*, volume 6056 of *Lecture Notes in Computer Science*, pages 420–443. Springer, 2010.

When Homomorphism Becomes a Liability

Zvika Brakerski*

Abstract

We show that an encryption scheme cannot have a simple decryption function and be homomorphic at the same time, even with added noise. Specifically, if a scheme can homomorphically evaluate the majority function, then its decryption cannot be weakly-learnable (in particular, linear), even if large decryption error is allowed. (In contrast, without homomorphism, such schemes do exist and are presumed secure, e.g. based on LPN.)

An immediate corollary is that known schemes that are based on the hardness of decoding in the presence of *low hamming-weight noise* cannot be fully homomorphic. This applies to known schemes such as LPN-based symmetric or public key encryption.

Using these techniques, we show that the recent candidate fully homomorphic encryption, suggested by Bogdanov and Lee (ePrint '11, henceforth BL), is insecure. In fact, we show two attacks on the BL scheme: One that uses homomorphism, and another that directly attacks a component of the scheme.

*Stanford University, zvika@stanford.edu. Supported by a Simons Postdoctoral Fellowship and by DARPA.

1 Introduction

An encryption scheme is called homomorphic if there is an efficient transformation that given $\text{Enc}(m)$ for some message m , and a function f , produces $\text{Enc}(f(m))$ using only public information. A scheme that is homomorphic w.r.t all efficient f is called fully homomorphic (FHE). Homomorphic encryption is a useful tool in both theory and practice and is extensively researched in recent years (see [Vai11] for survey), and a few candidates for full homomorphism are known.

Most of these candidates [Gen09, Gen10, SV10, BV11a, BV11b, GH11, BGV12, GHS12, Bra12] are based (either explicitly or implicitly) on *lattice assumptions* (the hardness of approximating short vectors in certain lattices). In particular, the learning with errors (LWE) assumption proved to be very useful in the design of such schemes. The one notable exception is [vDGHV10], but even that could be thought of as working over an appropriately defined lattice over the integers.

An important open problem is, therefore, to diversify and base fully homomorphic encryption on different assumptions (so as to not put all the eggs in one basket). One appealing direction is to try to use the learning parity with noise (LPN) problem, which is very similar in syntax to LWE: Making a vast generalization, LWE can be interpreted as a decoding problem for a linear code, where the noise comes from a family of *low norm* vectors. Namely, each coordinate in the code suffers from noise, but this noise is relatively small (this requires that the code is defined over a large alphabet). The LPN assumption works over the binary alphabet and requires that the noise has low hamming weight, namely that only a small number of coordinates are noisy, but in these coordinates the noise amplitude can be large. While similar in syntax, a direct connection between these two types of assumptions is not known.

While an LPN-based construction is not known, recently Bogdanov and Lee [BL11] presented a candidate, denoted by BL throughout this manuscript, that is based on a different low hamming-weight decoding problem: They consider a carefully crafted code over a large alphabet and assume that decoding in the presence of low-hamming-weight noise is hard.

In this work, we show that not only that BL's construction is insecure, but rather the entire approach of constructing code based homomorphic encryption analogously to the LWE construction cannot work. We stress that we don't show that FHE cannot be based on LPN (or other code based assumptions), but rather that the decryption algorithm of such scheme cannot take the naïve form. (In particular this applies to the attempt to add homomorphism to schemes such as [Ale03, GRS08, ACPS09].)

1.1 Our Results

Our main result shows that encryption schemes with learnable decryption functions cannot be homomorphic, even if large decryption error is allowed. In particular, such schemes cannot evaluate the majority function. This extends the result of Kearns and Valiant [KV94] (slightly extended by Klivans and Sherstov [KS09]) that learnability breaks security for schemes with negligible decryption error. In other words, homomorphic capabilities can sometimes make noisy learning become no harder than noiseless learning.

We use a simplified notion of learning, which essentially requires that given polynomially many labeled samples (from an arbitrary distribution), the learner's hypothesis correctly computes the label for the next sample with probability, say, 0.9. We show that this notion, that we call *sc-learning*, is equivalent to *weak learning* defined in [KV94]. This allows us to prove the following theorem (in Section 3).

Theorem A. *An encryption scheme whose decryption function is sc - or weakly-learnable, and whose decryption error is $1/2 - 1/\text{poly}(n)$, cannot homomorphically evaluate the majority function.*

Since it is straightforward to show that linear functions are learnable (as well as, e.g., low degree polynomials), the theorem applies to known LPN based schemes such as [Ale03, GRS08, ACPS09]. This may not seem obvious at first: The decryption circuit of the aforementioned schemes is (commonly assumed to be) hard to learn, and their decryption error is negligible, so they seem to be out of the scope of our theorem. However, looking more closely, the decryption circuits consist of an inner product computation with the secret key, followed by additional post-processing. One can verify that if the post processing is not performed, then correct decryption is still achieved with probability $> 1/2 + 1/\text{poly}$. Thus we can apply our theorem and rule out majority-homomorphism.

Similar logic rules out the homomorphism of the BL candidate-FHE. While Theorem A does not apply directly (since the decryption of BL is not learnable out of the box), we show that it contains a sub-scheme which is linear (and thus learnable) and has sufficient homomorphic properties to render it insecure.

Theorem B. *There is a successful polynomial time CPA attack on the BL scheme.*

We further present a different attack on the BL scheme, targeting one of its building blocks. This allows us to not only distinguish between two messages like the successful CPA attack above, but rather decrypt any ciphertext with probability $1 - o(1)$.

Theorem C. *There is a polynomial time algorithm that decrypts the BL scheme.*

The BL scheme and the two breaking algorithms are presented in Section 4.

1.2 Our Techniques

Consider a simplified case of Theorem A, where the scheme’s decryption function is learnable given t labeled samples, and the decryption error is (say) $1/(10(t + 1))$. The proof in this case is straightforward: Generate t labeled samples by just encrypting random messages. Then use the learner’s output hypothesis to decrypt the challenge ciphertext. We can only fail if either the learner fails (which happens with probability 0.1) or if one of the samples we draw (including the challenge) are not correctly decryptable, in which case our labeling is wrong and therefore the learner provides no guarantee (which again happens with at most 0.1 probability). The union bound implies that we can decrypt a random ciphertext with probability 0.8, which immediately breaks the scheme. Note that we did not use the homomorphism of the scheme at all, indeed this simplified version is universally true even without assuming homomorphism, and is very similar to the arguments in [KV94, KS09]. However, some subtleties arise since we allow a non-negligible fraction of “dysfunctional” keys that induce a much higher error rate than others.

The next step is to allow decryption error $1/2 - \epsilon$, which requires use of homomorphism. The idea is to use the homomorphism in order to reduce the decryption error and get back to the previous case (in other words, reducing the noise in a noisy learning problem). Consider a scheme that encrypts each message many times (say k), and then applies homomorphic majority on the ciphertexts. The security of this scheme directly reduces from that of the original scheme, and it has the same decryption function. However, now the decryption error drops exponentially with k . This is because in order to get an error in the new scheme, at least $k/2$ out of the k encryptions

need to have errors. Since the expected number is $(1/2 - \epsilon)k$, the Chernoff bound implies the result by choosing k appropriately.

To derive Theorem B, we need to show that linear functions are learnable:¹ Assume that the decryption function is an inner product between the ciphertext and the secret key (both being n -dimensional vectors over a field \mathbb{F}). We will learn these functions by taking $O(n)$ labeled samples. Then, given the challenge, we will try to represent it as a linear combination of the samples we have. If we succeed, then the appropriate linear combination of the labels will be the value of the function on the challenge. We show that this process fails only with small constant probability (intuitively, since we take $O(n)$ sample vectors from a space of dimension at most n).

We then show that BL uses a sub-structure that is both linearly decryptable and allows for homomorphism of (some sort of) majority. Theorem B thus follows similarly to Theorem A.

For Theorem C, we need to dive into the guts of the BL scheme. We notice that BL use homomorphic majority evaluation in one of the lower abstraction levels of their scheme. This allows us to break this abstraction level using only linear algebra (in a sense, the homomorphic evaluation is already “built in”). A complete break of BL follows.

1.3 Other Related Work

An independent work by Gauthier, Otmani and Tillich [GOT12] shows an interesting direct attack on BL’s hardness assumption (we refer to it as the “GOT attack”). Their attack is very different from ours and takes advantage of the resemblance of BL’s codes and Reed-Solomon codes as we explain below.

BL’s construction relies on a special type of error correcting code. Essentially, they start with a Reed-Solomon code, and replace a small fraction of the rows of the generating matrix with a special structure. The homomorphic properties are only due to this small fraction of “significant” rows, and the secret key is chosen so as to nullify the effect of the other rows.

The GOT attack uses the fact that under some transformation (component-wise multiplication), the dimension of Reed-Solomon codes can grow by at most a factor of two. However, if a code contains “significant” rows, then the dimension can grow further. This allows to measure the number of significant rows in a given code. One can thus identify the significant rows by trying to remove one row at a time from the code and checking if the dimension drops. If yes then that row is significant. Once all significant rows have been identified, the secret key can be retrieved in a straightforward manner.

However, it is fairly easy to immunize BL’s scheme against the GOT attack. As we explained above, the neutral rows do not change the properties of the encryption scheme, so they may as well be replaced by random rows. Since the dimension of random codes grows very rapidly under the GOT transformation, their attack will not work in such case.

Our attack, on the other hand, relies on certain functional properties that BL use to make their scheme homomorphic. Thus a change in the scheme that preserves these homomorphic properties cannot help to overcome our attack. In light of our attack, it is interesting to investigate whether the GOT attack can be extended to the more general case.

¹We believe this was known before, but since we could not find an appropriate reference, we provide a proof.

2 Preliminaries

We denote scalars using plain lowercase (x), vectors using bold lowercase (\mathbf{x} for column vector, \mathbf{x}^T for row vector), and matrices using bold uppercase (\mathbf{X}). We let $\mathbf{1}$ denote the all-one vector (the dimension will be clear from the context). We let \mathbb{F}_q denote a finite field of cardinality $q \in \mathbb{N}$, with efficient operations (we usually don't care about any other property of the field).

2.1 Properties of Encryption Schemes

A public key encryption scheme is a tuple of algorithms $(\text{Gen}, \text{Enc}, \text{Dec})$, such that: $\text{Gen}(1^n)$ is the key generation algorithm that produces a pair of public and secret keys (pk, sk) ; $\text{Enc}_{pk}(m)$ is a randomized encryption function that takes a message m and produces a ciphertext. In the context of this work, messages will only come from some predefined field \mathbb{F} ; $\text{Dec}_{sk}(c)$ is the decryption function that decrypts a ciphertext c and produces the message. Optimally, $\text{Dec}_{sk}(\text{Enc}_{pk}(\cdot))$ is the identity function, but in some schemes there are decryption errors.

The probability of decryption error is taken over the randomness used to generate the keys for the scheme, and over the randomness used in the encryption function (we assume the decryption is deterministic). Since in our case the error rates are high (approaching $1/2$), the effect of bad keys is different from that of bad encryption randomness, and we thus measure the two separately. We allow a small fraction of the keys (one percent, for the sake of convenience) to have arbitrarily large decryption error, and define the decryption error ϵ to be the maximal error over the 99% best keys. While the constant 1% is arbitrary and chosen so as to not over-clutter notation, we will discuss after presenting our results how they generalize to other values. The formal definition follows.

Definition 2.1. *An encryption scheme is said to have decryption error $< \epsilon$ if with probability at least 0.99 over the key generation it holds that*

$$\max_m \{\Pr[\text{Dec}_{sk}(\text{Enc}_{pk}(m)) \neq m]\} < \epsilon ,$$

where the probability is taken over the random coins of the encryption function.

We use the standard definition of security against chosen plaintext attacks (CPA): The attacker receives a public key and chooses two values m_0, m_1 . The attacker then receives a ciphertext $c = \text{Enc}_{pk}(m_b)$, where $b \in \{0, 1\}$ is a random bit that is unknown to the attacker. The attacker needs to decide on a guess $b' \in \{0, 1\}$ as to the value of b . We say that the scheme is broken if there is a polynomial time attacker for which $\Pr[b' = b] \geq 1/2 + 1/\text{poly}(n)$ (where n is the security parameter). Recall that this notion is equivalent to the notion of semantic security [GM82].

In addition, we will say that a scheme is *completely broken* if there exists an adversary that upon receiving the public key and $\text{Enc}_{pk}(m)$ for *arbitrary value of m* , returns m with probability $1 - o(1)$.

While we discuss homomorphic properties of encryption schemes, we will only use homomorphism w.r.t the majority function. We define the notion of k -majority-homomorphism below.

Definition 2.2. *A public-key encryption scheme is k -majority-homomorphic (where k is a function of the security parameter) if there exists a function MajEval such that with probability 0.99 over the key generation, for any sequence of ciphertexts output by $\text{Enc}_{pk}(\cdot)$: c_1, \dots, c_k , it holds that*

$$\text{Dec}_{sk}(\text{MajEval}_{pk}(c_1, \dots, c_k)) = \text{Majority}(\text{Dec}_{sk}(c_1), \dots, \text{Dec}_{sk}(c_k)) .$$

Again we allow some “slackness” by allowing some of the keys to not abide the homomorphism.

We note that Definition 2.2 above is a fairly strong notion of homomorphism in two aspects: First, it requires that homomorphism holds even for ciphertexts with decryption error. Second, we do not allow **MajEval** to introduce error for “good” key pairs. Indeed, known homomorphic encryption schemes have these properties, but it is interesting to try to bypass our negative results by finding schemes that do not have them.

Schemes with linear decryption, as defined below, have a special role in our attack on BL.

Definition 2.3. *An encryption scheme is n -linearly decryptable if its secret key is of the form $sk = \mathbf{s} \in \mathbb{F}^n$, for some field \mathbb{F} , and its decryption function is*

$$\text{Dec}_{sk}(\mathbf{c}) = \langle \mathbf{s}, \mathbf{c} \rangle .$$

2.2 Spanning Distributions over Low Dimensional Spaces

We will use a lemma that shows that any distribution over a low dimensional space is easy to span in the following sense: Given sufficiently many samples from the distribution (a little more than the dimension of the support), we are guaranteed that any new vector falls in the span of previous samples. This lemma will allow us to derive a (distribution-free) learner for linear functions (see Section 2.3).

We speculate that this lemma is already known, since it is fairly general and very robust to the definition of dimension (e.g. it also applies to non-linear spaces).

Lemma 2.1. *Let \mathcal{S} be a distribution over a linear space S of dimension s . For all k , define*

$$\delta_k \triangleq \Pr_{\mathbf{v}_1, \dots, \mathbf{v}_k \leftarrow \mathcal{S}} [\mathbf{v}_k \notin \text{Span} \{\mathbf{v}_1, \dots, \mathbf{v}_{k-1}\}] .$$

Then $\delta_k \leq s/k$.

Proof. Notice that by symmetry $\delta_i \geq \delta_{i+1}$ for all i . Let D_i denote the (random variable) dimension of $\text{Span} \{\mathbf{v}_1, \dots, \mathbf{v}_i\}$. Note that always $D_i \leq s$.

Let E_i denote the event $\mathbf{v}_i \notin \text{Span} \{\mathbf{v}_1, \dots, \mathbf{v}_{i-1}\}$, note that $\delta_i = \Pr[E_i]$. By definition,

$$D_k = \sum_{i=1}^k \mathbb{1}_{E_i} .$$

Therefore

$$s \geq \mathbb{E}[D_k] = \mathbb{E} \left[\sum_{i=1}^k \mathbb{1}_{E_i} \right] = \sum_{i=1}^k \Pr[E_i] = \sum_{i=1}^k \delta_i \geq k \cdot \delta_k ,$$

and the lemma follows. □

2.3 Learning

In this work we use two equivalent notions of learning: weak-learning as defined in [KV94], and an equivalent simplified notion that we call single-challenge-learning (sc-learning for short). The latter will be more convenient for our proofs, but we show that the two are equivalent. We will also show that linear functions are sc-learnable.

Notions of Learning. We start by introducing the notion of weak-learnability.

Definition 2.4 (weak-learning [KV94]). Let $\mathcal{F} = \{\mathcal{F}_n\}_{n \in \mathbb{N}}$ be an ensemble of binary functions. A weak learner for \mathcal{F} with parameters (t, ϵ, δ) is a polynomial time algorithm A such that for any function $f \in \mathcal{F}_n$ and for any distribution \mathcal{D} over the inputs to f , the following holds. Let $x_1, \dots, x_{t+1} \xleftarrow{\$} \mathcal{D}$, and let h (“the hypothesis”) be the output of $A(1^n, (x_1, f(x_1)), \dots, (x_t, f(x_t)))$. Then

$$\Pr_{x_1, \dots, x_t} \left[\Pr_{x_{t+1}} [h(x_{t+1}) \neq f(x_{t+1})] > \epsilon \right] \leq \delta .$$

We say that \mathcal{F} is weakly learnable if there exists a weak learner for \mathcal{F} with parameters $t = \text{poly}(n)$, $\epsilon \leq 1/2 - 1/\text{poly}(n)$, $\delta \leq 1 - 1/\text{poly}(n)$. (We also require that the output hypothesis h is polynomial time computable.)

We next define our notion of (t, η) -sc-learning, which essentially corresponds to the ability to launch a t -query CPA attack on an (errorless) encryption scheme, and succeed with probability η . (The initials “sc” stand for “single challenge”, reflecting the fact that a CPA attacker only receives a single challenge ciphertext.)

Definition 2.5 (sc-learning). Let $\mathcal{F} = \{\mathcal{F}_n\}_{n \in \mathbb{N}}$ be an ensemble of functions. A (t, η) -sc-learner for \mathcal{F} is a polynomial time algorithm A such that for any function $f \in \mathcal{F}_n$ and for any distribution \mathcal{D} over the inputs to f , the following holds. Let $x_1, \dots, x_{t+1} \xleftarrow{\$} \mathcal{D}$, and let h (“the hypothesis”) be the output of $A(1^n, (x_1, f(x_1)), \dots, (x_t, f(x_t)))$. Then $\Pr[h(x_{t+1}) \neq f(x_{t+1})] \leq \eta$, where the probability is taken over the entire experiment.

We say that \mathcal{F} is (t, η) -sc-learnable if it has a polynomial time (t, η) -sc-learner for it. We say that a binary \mathcal{F} is sc-learnable if $t = \text{poly}(n)$ and $\eta \leq 1/2 - 1/\text{poly}(n)$. (We also require that the output hypothesis h is polynomial time computable.)

Since sc-learning only involves one challenge, we do not define the “confidence” and “accuracy” parameters (δ, ϵ) separately as in the definition of weak-learning.

We note that both definitions allow for *improper learning* (namely, the hypothesis h does not need to “look like” an actual decryption function).

Equivalence Between Notions. The equivalence of the two notions is fairly straightforward. Applying boosting [Sch90] shows that sc-learning, like weak-learning, can be amplified.

Claim 2.2. If \mathcal{F} is sc-learnable then it is weak-learnable.

Proof. This follows by a Markov argument: Consider a (t, η) -sc-learner for \mathcal{F} (recall that $\eta \leq 1/2 - 1/\text{poly}(n)$) and let $\delta = 1 - 1/\text{poly}(n)$ be such that $\eta/\delta \leq 1/2 - 1/\text{poly}(n)$ (such δ must exist). Then letting $\epsilon \triangleq \eta/\delta$ finishes the argument. \square

The opposite direction will give us very strong amplification of learning by applying boosting.

Claim 2.3. If \mathcal{F} is weak-learnable then it is $(\text{poly}(n, 1/\eta), \eta)$ -sc-learnable for all η .

Proof. Let \mathcal{F} be weak-learnable. Then by boosting [Sch90] it is also PAC learnable [Val84]. Namely there is a learner with parameters $(\text{poly}(n, 1/\epsilon, 1/\delta), \epsilon, \delta)$ for any inversely polynomial ϵ, δ . Setting $\epsilon = \delta = \eta/2$, the claim follows. \square

Learning Linear Functions. The following corollary (of Lemma 2.1) shows an sc-learner for the class of linear functions.²

Corollary 2.4. *Let \mathcal{F}_n be a class of n -dimensional linear functions over a field \mathbb{F} . Then $\mathcal{F} = \{\mathcal{F}_n\}_n$ is $(10n, 1/10)$ -sc-learnable.*

Proof. The learner A is given $t = 10n$ samples $\mathbf{v}_i \triangleq (\mathbf{x}_i, f(\mathbf{x}_i)) \in \mathbb{F}^{n+1}$. Using Gaussian elimination, A will find $\mathbf{s} \in \mathbb{F}^n$ such that $(-\mathbf{s}, 1) \in \text{Ker}\{\mathbf{v}_i\}_{i \in [t]}$ (note that such must exist). Finally A will output the hypothesis $h(\mathbf{x}) = \langle \mathbf{s}, \mathbf{x} \rangle$.

Correctness follows using Lemma 2.1. We let the distribution \mathcal{S} be the distribution $(\mathbf{x}, f(\mathbf{x}))$ where $\mathbf{x} \xleftarrow{\$} \mathcal{D}$, and let $k = t+1$. Note that for any linear function $f : \mathbb{F}^n \rightarrow \mathbb{F}$, the set $\{(\mathbf{x}, f(\mathbf{x}))\}_{\mathbf{x} \in \mathbb{F}^n}$ is an n -dimensional linear subspace of \mathbb{F}^{n+1} .

Therefore, with probability $1 - 1/10$, it holds that $(\mathbf{x}_{t+1}, f(\mathbf{x}_{t+1})) \in \text{Span}\{\mathbf{v}_i\}_{i \in [t]}$ which implies that $\langle (-\mathbf{s}, 1), (\mathbf{x}_{t+1}, f(\mathbf{x}_{t+1})) \rangle = 0$, or in other words $f(\mathbf{x}_{t+1}) = \langle \mathbf{s}, \mathbf{x}_{t+1} \rangle = h(\mathbf{x}_{t+1})$. \square

3 Homomorphism is a Liability When Decryption is Learnable

This section features our main result. We show that schemes with learnable decryption circuits are very limited in terms of their homomorphic properties, regardless of decryption error. This extends the previous results of [KV94, KS09] showing that the decryption function cannot be learnable if the decryption error is negligible.

We start by showing that a scheme with $(t, 1/10)$ -sc-learnable decryption function (i.e. efficient learning with probability $1/10$ using t samples, see Definition 2.5) cannot have decryption error smaller than $\Omega(1/t)$ and be secure (regardless of homomorphism). We proceed to show that if the scheme can homomorphically evaluate the majority function, then the above amplifies dramatically and security cannot be guaranteed for *any* reasonable decryption error ($1/2 - \epsilon$ error for any noticeable ϵ). Using Claim 2.3 (boosting), this implies that the above hold for any scheme with weakly-learnable (or sc-learnable) decryption. We then discuss the role of key generation error compared to encryption error.

For the sake of simplicity, we focus on the public key setting. However, our proofs easily extend also to symmetric encryption, since our attacks only use the public key in order to generate ciphertexts for known messages.

Learnable Decryption without Homomorphism. We start by showing that a scheme whose decryption circuit is $(t, 1/10)$ -sc-learnable has to have decryption error $\epsilon = \Omega(1/t)$, otherwise it is insecure. This is a parameterized and slightly generalized version of the claims of [KV94, KS09], geared towards schemes with high decryption error and possibly bad keys. The basic idea is straightforward: We use the public key to generate t ciphertexts to be used as labeled samples for our learner, and then use its output hypothesis to decrypt the challenge ciphertext. The above succeeds so long as all samples in the experiment decrypt correctly, which by the union bound is at least $1 - t \cdot \epsilon$. A formal statement and proof follows.

Lemma 3.1. *An encryption scheme whose decryption function is $(t, 1/10)$ -sc-learnable for a polynomial t and whose decryption error $< 1/(10(t+1))$ is insecure.*

²The learner works even when the function class is not binary, which is only an advantage. The binary case follows by considering distributions supported only over the pre-images of 0, 1.

Proof. Consider a key pair (pk, sk) for the scheme, and consider the following CPA adversary. The adversary first generates t labeled samples of the form $(\text{Enc}_{pk}(m), m)$, for random messages $m \xleftarrow{\$} \{0, 1\}$ (where 0, 1 serve as generic names for an arbitrary pair of elements in the scheme's message space). These samples are fed into the aforementioned learner, let h denote the learner's output hypothesis. The adversary lets $m_0 = 0$, $m_1 = 1$, and given the challenge ciphertext $c = \text{Enc}_{pk}(m_b)$, it outputs $b' = h(c)$.

To analyze, we define \mathcal{D} to be the (inefficient) distribution $c = \text{Enc}_{pk}(m) | (\text{Dec}_{sk}(c) = m)$, for a randomly chosen $m \xleftarrow{\$} \{0, 1\}$. Namely, the distribution \mathcal{D} first samples $m \xleftarrow{\$} \{0, 1\}$, and then outputs a random *correctly decryptable* encryption of m . By Definition 2.5, if the learner gets t samples from this distribution, it outputs a hypothesis that correctly labels the $(t+1)$ sample, with all but $1/10$ probability.

While we cannot efficiently sample from \mathcal{D} (without the secret key), we show that the samples (and challenge) that we feed to our learner are in fact statistically close to samples from \mathcal{D} . Consider a case where (pk, sk) are such that the decryption error is indeed smaller than $\epsilon = 1/(10(t+1))$. In such case, our adversary samples from a distribution of statistical distance at most ϵ from \mathcal{D} , and the challenge ciphertext is drawn from the same distribution. It follows that the set of $(t+1)$ samples that we consider during the experiment (containing the labeled samples and the challenge), agree with \mathcal{D} with all but $(t+1) \cdot \epsilon = 1/10$ probability.

Using the union bound on all aforementioned “bad” events (the key pair not conforming with decryption error as per Definition 2.1, the samples not agreeing with \mathcal{D} , and the learner failing), we get that $\Pr[b' = b] \geq 1 - 0.01 - 1/10 - 1/10 > 0.7$ and the lemma follows. \square

Using Claim 2.3, we derive the following corollary.

Corollary 3.2. *An encryption scheme whose decryption function is weakly-learnable must have decryption error $1/\text{poly}(n)$ for some polynomial.*

We note that this corollary does not immediately follow from [KV94, KS09] if a noticeable fraction of the keys can be “bad” (since they do not use boosting).

Plugging our learner for linear functions (Corollary 2.4) into Lemma 3.1 implies the following, which will be useful for the next section.

Corollary 3.3. *There exists a constant $\alpha > 0$ such that any n -linearly decryptable scheme with decryption error $< \alpha/n$ is insecure.*

Learnable Decryption with Majority Homomorphism. Lemma 3.1 and Corollary 3.2 by themselves are not very restrictive. Specifically, they are not directly applicable to attacking any known scheme. Indeed, known schemes with linear decryption (e.g. LPN based) have sufficiently high decryption error (or, viewed differently, adding the error makes the underlying decryption hard to learn). We now show that if homomorphism is required as a property of the scheme, then decryption error cannot save us.

The following theorem states that majority-homomorphic schemes (see Definition 2.2) cannot have learnable decryption for any reasonable decryption error.

Theorem 3.4. *An encryption scheme whose decryption circuit is $(t, 1/10)$ -sc-learnable for a polynomial t and whose decryption error $< (1/2 - \epsilon)$ cannot be $O(\log t / \epsilon^2)$ -majority-homomorphic.*

Let us first outline the proof of Theorem 3.4 before formalizing it. Our goal is the same as in the proof of Lemma 3.1, to generate t labeled samples, which will enable to break security. However, unlike above, taking t random encryptions will surely introduce decryption errors. We thus use the majority homomorphism: We generate a good encryption of m , i.e. one that is decryptable with high probability, by generating $O(\log t/\epsilon^2)$ random encryptions of m , and apply majority homomorphically. Chernoff's bound guarantees that with high probability, more than half of the ciphertexts are properly decryptable, and therefore the output of the majority evaluation is with high probability a decryptable encryption of m . At this point, we can apply the same argument as in the proof of Lemma 3.1. The formal proof follows.

Proof. Consider an encryption scheme $(\text{Gen}, \text{Enc}, \text{Dec})$ as in the theorem statement. We will construct a new scheme $(\text{Gen}' = \text{Gen}, \text{Enc}', \text{Dec}' = \text{Dec})$ (with the same key generation and decryption algorithms) whose security relates to that of $(\text{Gen}, \text{Enc}, \text{Dec})$. Then we will use Lemma 3.1 to render the latter scheme insecure.

The new encryption algorithm $\text{Enc}'_{pk}(m)$ works as follows: To encrypt a message m , invoke the original encryption $\text{Enc}_{pk}(m)$ for (say) $k = 10(\ln(t+1) + \ln(10))/\epsilon^2$ times, thus generating k ciphertexts. Apply **MajEval** to those k ciphertexts and output the resulting ciphertext.

The security of the new scheme is related to that of the original by a straightforward hybrid argument. We will show that the new scheme has decryption error at most $1/(10(t+1))$, but in a slightly weaker sense than Definition 2.1: We will allow 2% of the keys to be “bad” instead of just 1% as before. One can easily verify that the proof of Lemma 3.1 works in this case as well.

Our set of good key pairs for Enc' is those for which $\text{Dec}_{sk}(\text{Enc}_{pk}(\cdot))$ indeed have decryption error at most $1/2 - \epsilon$ and in addition **MajEval** is correct. By the union bound this happens with probability at least 0.98.

To bound the decryption error of $\text{Dec}_{sk}(\text{Enc}'_{pk}(\cdot))$, assume that we have a good key pair as described above. We will bound the probability that more than a $1/2 - \epsilon/2$ fraction of the k ciphertexts generated by Enc' are decrypted incorrectly. Clearly if this bad event does not happen, then by the correctness of **MajEval**, the resulting ciphertext will decrypt correctly.

Recalling that the expected fraction of falsely decrypted ciphertexts is at most $1/2 - \epsilon$, the Chernoff bound implies that the aforementioned bad event happens with probability at most

$$e^{-2(\epsilon/2)^2 k} < 1/(10(t+1)) ,$$

and the theorem follows. \square

From the proof it is obvious that even “approximate-majority homomorphism” is sufficient for the theorem to hold. Namely, even if **MajEval** only computes the majority function correctly if the fraction of identical inputs is more than $1/2 + \epsilon/2$.

We can derive a general corollary for every weakly-learnable function using Claim 2.3. This applies, for example, to linear functions, low degree polynomials and shallow circuits.

Corollary 3.5. *An encryption scheme whose decryption function is weakly-learnable and whose decryption error is $1/2 - \epsilon$ cannot be $\omega(\log n/\epsilon^2)$ -majority-homomorphic.*

The Role of Bad Keys. Recall that in Definitions 2.1 and 2.2 (decryption error and majority homomorphism) we allowed a constant fraction of keys to be useless for the purpose of decryption

and homomorphic evaluation, respectively. In fact, it is this relaxation that makes our argument more involved than [KV94, KS09].

As we mentioned above, the choice of constant 0.01 is arbitrary. Let us now explain how our results extend to the case of $1/2 - \kappa$ fraction of bad keys, where $\kappa = 1/\text{poly}(n)$ (we now count the keys that are either bad for decryption or bad for homomorphism). In such case, the argument of Lemma 3.1 will work so long as we start with a (t, η) -sc-learner with $\eta < \kappa/3$ and so long as the decryption error for good keys is at most $\kappa/(3(t+1))$. If the scheme is furthermore $O(\log(t/\kappa)/\epsilon^2) = O(\log n/\epsilon^2)$ -majority-homomorphic, the proof of Theorem 3.4 will also go through. Finally, using boosting, we can start with any weak learner and reduce η to $< \kappa/3$ at the cost of a polynomial increase in t , which is tolerable by our arguments (and swallowed by the asymptotic notation).

4 Attacks on the BL Scheme

In this section we use our tools from above to show that the BL scheme (outlined in Section 4.1 below) is broken. We present two attacks: the first, in Section 4.2, follows from Corollary 3.3 (and works in the spirit of Theorem 3.4); and the second, in Section 4.3, directly attacks a lower level subcomponent of the scheme and allows to decrypt any ciphertext. In fact, the latter attack also follows the same basic principles and exploits a “built-in” evaluation of majority that exists in that sub-component of BL.

4.1 Essentials of the BL Scheme

In this section we present the properties of the BL scheme. We concentrate on the properties that are required for our breaks. We refer the reader to [BL11] for further details.

The BL scheme has a number of layers of abstraction, which are all instantiated based on a global parameter $0 < \alpha < 0.25$ as explained below.

The Scheme $\mathbf{K}_q(n)$. BL introduce $\mathbf{K}_q(n)$, a public-key encryption scheme with imperfect correctness. For security parameter n , the public key is a matrix $\mathbf{P} \in \mathbb{F}_q^{n \times r}$, where $r = n^{1-\alpha/8}$, and the secret key is a vector $\mathbf{y} \in \mathbb{F}_q^n$ in the kernel of \mathbf{P}^T (namely, $\mathbf{y}^T \cdot \mathbf{P} = 0$). The keys are generated in a highly structured manner in order to support homomorphism, but their structure is irrelevant to us. An encryption of a message $m \in \mathbb{F}_q$ is a vector $\mathbf{c} = \mathbf{P} \cdot \mathbf{x} + m \cdot \mathbf{1} + \mathbf{e}$, where $\mathbf{x} \in \mathbb{F}_q^r$ is some vector, and where $\mathbf{e} \in \mathbb{F}_q^n$ is a low hamming weight vector. Decryption is performed by taking the inner product $\langle \mathbf{y}, \mathbf{c} \rangle$, and succeeds so long as $\langle \mathbf{y}, \mathbf{e} \rangle = 0$ (the vector \mathbf{y} is chosen such that $\langle \mathbf{y}, \mathbf{1} \rangle = 1$). It is shown how the structure of the keys implies that decryption succeeds with probability at least $(1 - n^{-(1-\alpha/2)})$. Finally, BL show that $\mathbf{K}_q(n)$ is homomorphic with respect to a single addition or multiplication.³

Re-Encryption. In order to enable homomorphism, BL introduce the notion of re-encryption. Consider an instantiation of $\mathbf{K}_q(n)$, with keys (\mathbf{P}, \mathbf{y}) , and an instantiation of $\mathbf{K}_q(n')$ with keys $(\mathbf{P}', \mathbf{y}')$, for $n' = n^{1+\alpha}$. Let $\mathbf{H}_{n':n} \in \mathbb{F}_q^{n' \times n}$ be an element-wise encryption of \mathbf{y} using the public key

³Homomorphic operations (addition, multiplication) are performed element-wise on ciphertext vectors, and the structure of the key guarantees that correctness is preserved.

\mathbf{P}' .⁴ Namely $\mathbf{H}_{n':n} = \mathbf{P}' \cdot \mathbf{X}' + \mathbf{1} \cdot \mathbf{y}^T + \mathbf{E}'$. Due to the size difference between the schemes, it holds that with probability $(1 - n^{-\Omega(1)})$, all of the columns of $\mathbf{H}_{n':n}$ are simultaneously decryptable and indeed $\mathbf{y}^T \cdot \mathbf{H}_{n':n} = \mathbf{y}^T$. In such case, for any ciphertext \mathbf{c} of $\mathbf{K}_q(n)$, we get $\langle \mathbf{y}', \mathbf{H}_{n':n} \mathbf{c} \rangle = \langle \mathbf{y}, \mathbf{c} \rangle$. The matrix $\mathbf{H}_{n':n}$ therefore re-encrypts ciphertexts of $\mathbf{K}_q(n)$ as ciphertexts of $\mathbf{K}_q(n')$.

The critical idea for our second break is that a re-encrypted ciphertext always belongs to an n -dimensional linear subspace (recall that $n \ll n'$), namely to the span of $\mathbf{H}_{n':n}$.

The Scheme BASIC. Using re-encryption, BL construct a ladder of schemes of increasing lengths that allow for homomorphic evaluation. They define the scheme **BASIC** which has an additional depth parameter $d = O(1)$ (BL suggest to use $d = 8$, but our attack works for any $d > 1$). They consider instantiations of $\mathbf{K}_q(n_i)$, where $n_i = n^{(1+\alpha)^{-(d-i)}}$, for $i = 0, \dots, d$, so $n_d = n$. They generate all re-encryption matrices $\mathbf{H}_{n_{i+1}:n_i}$ (with success probability $(1 - n^{-\Omega(1)})$) and can thus homomorphically evaluate depth d circuits.

The homomorphic evaluation works by performing a homomorphic operation at level i of the evaluated circuit (with i going from 0 to $d - 1$), and then using re-encryption with $\mathbf{H}_{n_{i+1}:n_i}$ to obtain a fresh ciphertext for the next level.

For the purposes of our (second) break, we notice that in the last step of this evaluation is re-encryption using $\mathbf{H}_{n_d:n_{d-1}}$. This means that homomorphically evaluated ciphertexts all come from a linear subspace of dimension $n_{d-1} = n^{1/(1+\alpha)}$.

Error Correction and the Matrix $\mathbf{H}_{n:n}$. The scheme **BASIC** only allows homomorphism at the expense of increasing the instance size (namely n). BL show next that it is possible to use **BASIC** to generate a re-encryption matrix without a size increase.

They generate an instance of **BASIC**, with public keys $\mathbf{P}_0, \dots, \mathbf{P}_d$, secret key $\mathbf{y}_d = \mathbf{y}^*$, and re-encryption matrices $\mathbf{H}_{n_{i+1}:n_i}$. An additional independent instance of $\mathbf{K}_q(n)$ is generated, whose keys we denote by (\mathbf{P}, \mathbf{y}) . Then, a large number of encryptions of the elements of \mathbf{y} under public key \mathbf{P}_0 are generated.⁵ While some of these ciphertexts may have encryption error, BL show that homomorphically evaluating a depth- d correction circuit (*CORR* in their notation), one can obtain a matrix $\mathbf{H}_{n:n}$, whose columns are encryptions of \mathbf{y}^* that are decryptable under \mathbf{y} without error. This process succeeds with probability $(1 - n^{-\Omega(1)})$.

The resemblance to the learner of Corollary 2.4 is apparent. In a sense, the public key of **BASIC** is ready-for-use learner.

To conclude this part, BL generate a re-encryption matrix $\mathbf{H}_{n:n}$ that takes ciphertexts under \mathbf{y} and produces ciphertexts under \mathbf{y}^* . Since $\mathbf{H}_{n:n}$ is produced using homomorphic evaluation, its rank is at most $n_{d-1} = n^{1/(1+\alpha)}$. We will capitalize on the fact that re-encryption using $\mathbf{H}_{n:n}$ produces ciphertexts that all reside in a low-dimensional space.

Achieving Full Homomorphism – The Scheme HOM. The basic idea is to generate a sequence of matrices $\mathbf{H}_{n:n}$, thus creating a chaining of the respective secret keys that will allow homomorphism of any depth. However, generating an arbitrarily large number of such re-encryption matrices will eventually cause an error somewhere down the line. Therefore, a more sophisticated

⁴A note on notation: In [BL11], the re-encryption parameters are denoted by I (as opposed to our \mathbf{H}). We feel that their notation ignores the important linear algebraic structure of the re-encryption parameters, and therefore we switched to matrix notation, which also dictated the change of letter.

⁵To be absolutely precise, BL encrypt a bit decomposition of \mathbf{y}^* , but this is immaterial to us.

solution is required. BL suggest to encrypt each message a large number of times, and generate a large number of re-encryption matrices per level. Then, since the vast majority of matrices per level are guaranteed to be correct, one can use shallow approximate majority computation to guarantee that the fraction of erroneous ciphertexts per level does not increase with homomorphic evaluation.

Decryption is performed as follows: Each ciphertext is a set of ciphertexts c_1, \dots, c_k of $\mathbf{K}_q(n)$ (all with the same secret key). The decryption process first uses the $\mathbf{K}_q(n)$ key to decrypt the individual ciphertexts and obtain m_1, \dots, m_k , and then outputs the majority between the values m_i . BL show that a majority of the ciphertexts (say more than 15/16 fraction) are indeed correct, which guarantees correct decryption.

BL can thus achieve a (leveled) fully homomorphic scheme which they denote by **HOM**, which completes their construction.

4.2 An Attack on BL Using Homomorphism

We will show how to break the BL scheme using its homomorphic properties. We use Corollary 3.3 and our proof contains similar elements to the proof of Theorem 3.4. (The specifics of BL do not allow to use Corollary 3.5 directly.)

Theorem 4.1. *There is a polynomial time CPA attack on BL.*

Proof. Clearly we cannot apply our methods to the scheme **HOM** as is, since its decryption is not learnable. We thus describe a related scheme which is “embedded” in **HOM** and show how to distinguish encryptions of 0 from encryptions of 1, which will imply a break of **HOM**.

We recall that the public key of **HOM** contains “chains” of re-encryption matrices of the form $\mathbf{H}_{n:n}$. The length of the chains is related to the homomorphic depth of **HOM**. Our sub-scheme will only require a chain of constant length ℓ which will be determined later (such sub-chain therefore must exist for any instantiation of BL that allows for more than constant depth homomorphism). Granted that all links in the chain are successfully generated (which happens with probability $\ell \cdot n^{-\Omega(1)}$), such a chain allows homomorphic evaluation of any depth- ℓ function. Let us focus on the case where the chain is indeed properly generated.

Intuitively, we would have liked to use this structure to evaluate majority on 2^ℓ input ciphertexts. However, BL is defined over a large field \mathbb{F} , and it is not clear how to implement majority over \mathbb{F} in depth that does not depend on $q = |\mathbb{F}|$. To solve this problem, we use BL’s *CORR* function. This function is just a NAND tree of depth ℓ (extended to \mathbb{F} in the obvious way: $\text{NAND}(x, y) = 1 - xy$). BL show that given 2^ℓ inputs, each of which is 0 (respectively 1) with probability $1 - \epsilon$, the output of *CORR* will be 0 (resp. 1) with probability $1 - O(\epsilon)^{2^{\ell/2}}$.

To encrypt a message $m \in \{0, 1\}$ using our sub-scheme, we will generate 2^ℓ ciphertexts. Each ciphertext will be an independent encryption of m using the public key of **HOM** (which essentially generates $\mathbf{K}_q(n)$ ciphertexts that correspond to the first link in all chains). We then apply *CORR* homomorphically to the generated ciphertexts. Decryption in our subscheme will be standard $\mathbf{K}_q(n)$ decryption (which is a linear function) using the secret key that corresponds to the last link in the chain.⁶

We recall that the decryption error of $\mathbf{K}_q(n)$ is $\epsilon = n^{-\Omega(1)}$. By the properties of *CORR*, we can choose $\ell = O(1)$ such that the decryption error of our sub-scheme is at most (say) $o(1)/n$.

⁶The secret key of the last link is not the same as the secret key of **HOM**, since we are only considering a sub-chain of a much longer chain. However, this is not a problem: Our arguments do not require that the secret key is known to anyone in order to break the scheme.

In conclusion, we get a sub-scheme of **HOM** such that with probability $1 - n^{-\Omega(1)} > 0.9$ over the key generation, the decryption error is at most $o(1)/n$. Furthermore, decryption is linear. Corollary 3.3 implies that such scheme must be insecure. \square

4.3 A Specific Attack on BL

We noticed that the scheme **BASIC**, which is a component of **HOM**, contains by design homomorphic evaluation of majority: this is how the matrix $\mathbf{H}_{n:n}$ is generated. We thus present an attack that only uses the matrix $\mathbf{H}_{n:n}$ and allows to completely decrypt BL ciphertexts (even non binary) with probability $1 - n^{-\Omega(1)}$. We recall that an attack *completely breaks* a scheme if it can decrypt any given ciphertext with probability $1 - o(1)$.

Theorem 4.2. *There exists a polynomial time attack that completely breaks **BASIC**, and thus also BL.*

Proof. We consider the re-encryption matrix $\mathbf{H} = \mathbf{H}_{n:n} \in \mathbb{F}_q^{n \times n}$ described in Section 4.1, which re-encrypts ciphertexts under \mathbf{y} into ciphertexts under \mathbf{y}^* . The probability that \mathbf{H} was successfully generated is at least $1 - n^{-\Omega(1)}$, in which case it holds that

$$\mathbf{y}^{*T} \cdot \mathbf{H} = \mathbf{y}^T.$$

In addition, as we explained in Section 4.1, the rank of \mathbf{H} is at most $h = n^{1/(1+\alpha)}$.

Our breaker will be given \mathbf{H} and the public key \mathbf{P} that corresponds to \mathbf{y} , and will be able to decrypt any vector $\mathbf{c} = \text{Enc}_{\mathbf{P}}(m)$ with high probability, namely compute $\langle \mathbf{y}, \mathbf{c} \rangle$.

Breaker Code. As explained above, the input to the breaker is \mathbf{H}, \mathbf{P} and challenge $\mathbf{c} = \text{Enc}_{\mathbf{P}}(m)$. The breaker will execute as follows:

1. Generate $k = h^{1+\epsilon}$ encryptions of 0, denoted $\mathbf{v}_1, \dots, \mathbf{v}_k$, for $\epsilon = \frac{\alpha(1-\alpha)}{4}$ (any positive number smaller than $\frac{\alpha(1-\alpha)}{2}$ will do).

Note that this means that with probability $1 - n^{-\Omega(1)}$, all \mathbf{v}_i are decryptable encryptions of 0. Intuitively, these vectors, once projected through \mathbf{H} , will span all decryptable encryptions of 0.

2. For all $i = 1, \dots, k$, compute $\mathbf{v}_i^* = \mathbf{H} \cdot \mathbf{v}_i$ (the projections of the ciphertexts above through \mathbf{H}). Also compute $\mathbf{o}^* = \mathbf{H} \cdot \mathbf{1}$ (the projection of the all-one vector).
3. Find a vector $\tilde{\mathbf{y}}^* \in \mathbb{F}_q^n$ such that $\langle \tilde{\mathbf{y}}^*, \mathbf{v}_i^* \rangle = 0$ for all i , and such that $\langle \tilde{\mathbf{y}}^*, \mathbf{o}^* \rangle = 1$. Such a vector necessarily exists if all \mathbf{v}_i 's are decryptable, since \mathbf{y}^* is an example of such a vector.
4. Given a challenge ciphertext \mathbf{c} , compute $\mathbf{c}^* = \mathbf{H} \cdot \mathbf{c}$ and output $m = \langle \tilde{\mathbf{y}}^*, \mathbf{c}^* \rangle$ (namely, $m = \tilde{\mathbf{y}}^{*T} \cdot \mathbf{H} \cdot \mathbf{c}$).

Correctness. To analyze the correctness of the breaker, we first notice that the space of ciphertexts that decrypt to 0 under \mathbf{y} is linear (this is exactly the orthogonal space to \mathbf{y}). We denote this space by Z . Since $\mathbf{1} \notin Z$, we can define the cosets $Z_m = Z + m \cdot \mathbf{1}$. We note that all legal encryptions of m using \mathbf{P} reside in Z_m .

We let Z^* denote the space $\mathbf{H} \cdot Z$ (all vectors of the form $\mathbf{H} \cdot \mathbf{z}$ such that $\mathbf{z} \in Z$). This is a linear space with dimension at most h . Similarly, define $Z_m^* = Z^* + m \cdot \mathbf{o}^*$.

Consider the challenge ciphertext $\mathbf{c} = \text{Enc}_{\mathbf{P}}(m)$. We can think of \mathbf{c} as an encryption of 0 with an added term $m \cdot \mathbf{1}$. We therefore denote $\mathbf{c} = \mathbf{c}_0 + m \cdot \mathbf{1}$. Again this yields a \mathbf{c}_0^* such that $\mathbf{c}^* = \mathbf{c}_0^* + m \cdot \mathbf{o}^*$.

Now consider the distribution \mathcal{Z} over Z , which is the distribution of decryptable encryptions of 0 (i.e. the distribution $\mathbf{c} = \text{Enc}_{\mathbf{P}}(0)$, conditioned on $\langle \mathbf{y}, \mathbf{c} \rangle = 0$). The distribution \mathcal{Z}^* is defined by projecting \mathcal{Z} through \mathbf{H} . With probability $(1 - n^{-\Omega(1)})$, it holds that $\mathbf{v}_1^*, \dots, \mathbf{v}_k^*$, and \mathbf{c}_0^* are uniform samples from \mathcal{Z}^* .

By Lemma 2.1 below, it holds that $\mathbf{c}_0^* \in \text{Span}\{\mathbf{v}_1^*, \dots, \mathbf{v}_k^*\}$, with probability $(1 - n^{-\Omega(1)})$. In such case

$$\langle \tilde{\mathbf{y}}^*, \mathbf{c}^* \rangle = \langle \tilde{\mathbf{y}}^*, \mathbf{c}_0^* \rangle + m \cdot \langle \tilde{\mathbf{y}}^*, \mathbf{o}^* \rangle = m.$$

We conclude that with probability $1 - n^{-\Omega(1)}$, our breaker correctly decrypts \mathbf{c} as required. \square

Acknowledgements

The author wishes to thank the MIT-BU cryptography reading group for introducing the BL scheme to him, and especially to Stefano Tessaro and Shafi Goldwasser for various discussions. We further thank Andrej Bogdanov for his comments on a preprint of this manuscript, and for the discussions that followed, leading to the general argument about homomorphic schemes. Lastly, we thank Ron Rothblum for pointing out the learning-theory aspect of our argument.

References

- [ACPS09] Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In Shai Halevi, editor, *CRYPTO*, volume 5677 of *Lecture Notes in Computer Science*, pages 595–618. Springer, 2009.
- [Ale03] Michael Alekhnovich. More on average case vs approximation complexity. In *FOCS*, pages 298–307. IEEE Computer Society, 2003.
- [BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ITCS*, 2012. See also <http://eprint.iacr.org/2011/277>.
- [BL11] Andrej Bogdanov and Chin Ho Lee. Homomorphic encryption from codes. *Cryptology ePrint Archive*, Report 2011/622, 2011. <http://eprint.iacr.org/>.
- [Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO*, volume 7417 of *Lecture Notes in Computer Science*, pages 868–886. Springer, 2012. See also <http://eprint.iacr.org/2012/078>.
- [BV11a] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-LWE and security for key dependent messages. In *CRYPTO*, volume 6841, page 501, 2011.

- [BV11b] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In Rafail Ostrovsky, editor, *FOCS*, pages 97–106. IEEE, 2011. References are to full version: <http://eprint.iacr.org/2011/344>.
- [Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009.
- [Gen10] Craig Gentry. Toward basing fully homomorphic encryption on worst-case hardness. In *CRYPTO*, pages 116–137, 2010.
- [GH11] Craig Gentry and Shai Halevi. Implementing gentry’s fully-homomorphic encryption scheme. In Kenneth G. Paterson, editor, *EUROCRYPT*, volume 6632 of *Lecture Notes in Computer Science*, pages 129–148. Springer, 2011.
- [GHS12] Craig Gentry, Shai Halevi, and Nigel P. Smart. Fully homomorphic encryption with polylog overhead. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT*, volume 7237 of *Lecture Notes in Computer Science*, pages 465–482. Springer, 2012.
- [GM82] Shafi Goldwasser and Silvio Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In Harry R. Lewis, Barbara B. Simons, Walter A. Burkhard, and Lawrence H. Landweber, editors, *STOC*, pages 365–377. ACM, 1982.
- [GOT12] Valérie Gauthier, Ayoub Otmani, and Jean-Pierre Tillich. A distinguisher-based attack of a homomorphic encryption scheme relying on reed-solomon codes. Cryptology ePrint Archive, Report 2012/168, 2012. <http://eprint.iacr.org/>.
- [GRS08] Henri Gilbert, Matthew J. B. Robshaw, and Yannick Seurin. How to encrypt with the lpn problem. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfssdóttir, and Igor Walukiewicz, editors, *ICALP (2)*, volume 5126 of *Lecture Notes in Computer Science*, pages 679–690. Springer, 2008.
- [KS09] Adam R. Klivans and Alexander A. Sherstov. Cryptographic hardness for learning intersections of halfspaces. *J. Comput. Syst. Sci.*, 75(1):2–12, 2009. Preliminary version in FOCS 06.
- [KV94] Michael J. Kearns and Leslie G. Valiant. Cryptographic limitations on learning boolean formulae and finite automata. *J. ACM*, 41(1):67–95, 1994. Preliminary version in STOC 89.
- [Sch90] Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 5:197–227, 1990. Preliminary version in FOCS 89.
- [SV10] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In Phong Q. Nguyen and David Pointcheval, editors, *Public Key Cryptography*, volume 6056 of *Lecture Notes in Computer Science*, pages 420–443. Springer, 2010.

- [Vai11] Vinod Vaikuntanathan. Computing blindfolded: New developments in fully homomorphic encryption. In Rafail Ostrovsky, editor, *FOCS*, pages 5–16. IEEE, 2011.
- [Val84] Leslie G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, 1984. Preliminary version in STOC 84.
- [vDGHV10] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *EUROCRYPT*, pages 24–43, 2010. Full Version in <http://eprint.iacr.org/2009/616.pdf>.

Quantum-Secure Message Authentication Codes

Dan Boneh Mark Zhandry

Stanford University
{dabo,zhandry}@cs.stanford.edu

Abstract

We construct the first Message Authentication Codes (MACs) that are existentially unforgeable against a *quantum* chosen message attack. These chosen message attacks model a quantum adversary’s ability to obtain the MAC on a superposition of messages of its choice. We begin by showing that a quantum secure PRF is sufficient for constructing a quantum secure MAC, a fact that is considerably harder to prove than its classical analogue. Next, we show that a variant of Carter-Wegman MACs can be proven to be quantum secure. Unlike the classical settings, we present an attack showing that a pair-wise independent hash family is insufficient to construct a quantum secure *one-time* MAC, but we prove that a four-wise independent family is sufficient for one-time security.

Keywords: Quantum computing, MAC, chosen message attacks, post-quantum security

1 Introduction

Message Authentication Codes (MACs) are an important building block in cryptography used to ensure data integrity. A MAC system is said to be secure if an efficient attacker capable of mounting a chosen message attack cannot produce an existential MAC forgery (see Section 2.2).

With the advent of quantum computing there is a strong interest in post-quantum cryptography, that is systems that remain secure even when the adversary has access to a quantum computer. There are two natural approaches to defining security of a MAC system against a quantum adversary. One approach is to restrict the adversary to issue classical chosen message queries, but then allow the adversary to perform quantum computations between queries. Security in this model can be achieved by basing the MAC construction on a quantum intractable problem.

The other more conservative approach to defining quantum MAC security is to model the entire security game as a quantum experiment and allow the adversary to issue *quantum* chosen message queries. That is, the adversary can submit a superposition of messages from the message space and in response receive a superposition of MAC tags on those messages. Informally, a quantum chosen message query performs the following transformation on a given superposition of messages:

$$\sum_m \psi_m |m\rangle \longrightarrow \sum_m \psi_m |m, S(k, m)\rangle$$

where $S(k, m)$ is a tag on the message m with secret key k .

To define security, let q be the number of queries that the adversary issues by the end of the game. Clearly it can now produce q classical message-tag pairs by sampling the q superpositions

it received from the MAC signing oracle. We say that the MAC system is *quantum secure* if the adversary cannot produce $q + 1$ valid message-tag pairs. This captures the fact that the adversary cannot do any better than trivially sampling the responses from its MAC signing oracle and is the quantum analogue of a classical existential forgery.

1.1 Our results

In this paper we construct the first quantum secure MAC systems. We begin with a definition of quantum secure MACs and give an example of a MAC system that is secure against quantum adversaries capable of *classical* chosen message queries, but is insecure when the adversary can issue *quantum* chosen message queries. We then present a number of quantum secure MAC systems.

Quantum secure MACs. In the classical settings many MAC systems are based on the observation that a secure pseudorandom function gives rise to a secure MAC [BKR00, BCK96]. We begin by studying the same question in the quantum settings. Very recently Zhandry [Zha12b] defined the concept of a quantum secure pseudorandom function (PRF) which is a PRF that remains indistinguishable from a random function even when the adversary can issue *quantum* queries to the PRF. He showed that the classic GGM construction [GGM86] remains secure under quantum queries assuming the underlying pseudorandom generator is quantum secure.

The first question we study is whether a quantum secure PRF gives rise to a quantum secure MAC, as in the classical settings. To the MAC adversary a quantum secure PRF is indistinguishable from a random function. Therefore proving that the MAC is secure amounts to proving that with q quantum queries to a random oracle H no adversary can produce $q + 1$ input-output pairs of H with non-negligible probability. In the classical settings where the adversary can only issue *classical* queries to H this is trivial: given q evaluations of a random function, the adversary learns nothing about the value of the function at other points. Unfortunately, this argument fails under quantum queries because the response to a *single* quantum query to $H : \mathcal{X} \rightarrow \mathcal{Y}$ contains information about all of H . In fact, with a single quantum query the adversary can produce two input-output pairs of H with probability about $2/|\mathcal{Y}|$ (with classical queries the best possible is $1/|\mathcal{Y}|$). As a result, proving that q quantum queries are insufficient to produce $q + 1$ input-output pairs is quite challenging. We prove tight upper and lower bounds on this question by proving the following theorem:

Theorem 1.1 (informal). *Let $H : \mathcal{X} \rightarrow \mathcal{Y}$ be a random oracle. Then an adversary making at most $q < |\mathcal{X}|$ quantum queries to H will produce $q + 1$ input-output pairs of H with probability at most $(q + 1)/|\mathcal{Y}|$. Furthermore, when $q \ll |\mathcal{Y}|$ there is an algorithm that with q quantum queries to H will produce $q + 1$ input-output pairs of H with probability $1 - (1 - 1/|\mathcal{Y}|)^{q+1} \approx (q + 1)/|\mathcal{Y}|$.*

The first part of the theorem is the crucial fact needed to build quantum secure MACs and is the harder part to prove. It shows that when $|\mathcal{Y}|$ is large any algorithm has only a negligible chance in producing $q + 1$ input-output pairs of H from q quantum queries. To prove this bound we introduce a new lower-bound technique we call the *rank method* for bounding the success probability of algorithms that succeed with only small probability. Existing quantum lower bound techniques such as the polynomial method [BBC⁺01] and the adversary method [Amb00, Aar02, Amb06, ASdW09] do not give the result we need. One difficulty with existing lower bound techniques is that they generally prove asymptotic bounds on the number of queries required to solve a problem with high probability, whereas we need a bound on the success probability of an algorithm making a limited number of queries. Attempting to apply existing techniques to our problem at best only bounds

the success probability away from 1 by an inverse polynomial factor, which is insufficient for our purposes. The rank method for proving quantum lower bounds overcomes these difficulties and is a general tool that can be used in other post-quantum security proofs.

The second part of Theorem 1.1 shows that the lower bound presented in the first part of the theorem is tight. A related algorithm was previously presented by van Dam [vD98], but only for oracles outputting one bit, namely when $\mathcal{Y} = \{0, 1\}$. For such a small range only about $|\mathcal{X}|/2$ quantum queries are needed to learn the oracle at all $|\mathcal{X}|$ points. A special case where $\mathcal{Y} = \mathcal{X} = \{0, 1\}$ and $q = 1$ was developed independently by Kerenidis and de Wolf [KdW03]. Our algorithm is a generalization of van Dam’s result to multi-bit oracles.

Quantum secure Carter-Wegman MACs. A Carter-Wegman MAC [WC81] signs a message m by computing $(r, h(m) \oplus F(k, r))$ where h is a secret hash function chosen from an xor-universal hash family, F is a secure PRF with secret key k , and r is a short random nonce. The attractive feature of Carter-Wegman MACs is that the long message m is hashed by a fast xor-universal hash h . We show that a slightly modified Carter-Wegman MAC is quantum secure assuming the underlying PRF is quantum secure in the sense of Zhandry [Zha12b].

One-time quantum secure MACs. A one-time MAC is existentially unforgeable when the adversary can make only a *single* chosen message query. Classically, one-time MACs are constructed from pair-wise independent hash functions [WC81]. These MACs are one-time secure since the value of a pair-wise independent hash at one point gives no information about its value at another point. Therefore, a single classical chosen-message query tells the adversary nothing about the MAC tag of another message.

In the quantum settings things are more complicated. Unlike the classical settings, we show that pair-wise independence does not imply existential unforgeability under a one-time quantum chosen message attack. For example, consider the simple pair-wise independent hash family $\mathcal{H} = \{h(x) = ax + b\}_{a,b \in \mathbb{F}_p}$ with domain and range \mathbb{F}_p . We show that a quantum adversary presented with an oracle for a random function $h \in \mathcal{H}$ can find both a and b with a *single* quantum query to h . Consequently, the classical one-time MAC constructed from \mathcal{H} is completely insecure in the quantum settings. More generally we prove the following theorem:

Theorem 1.2 (informal). *There is a polynomial time quantum algorithm that when presented with an oracle for $h(x) = a_0 + a_1x + \dots + a_dx^d$ for random a_0, \dots, a_d in \mathbb{F}_p can recover a_0, \dots, a_d using only d quantum queries to the oracle with probability $1 - O(d/n)$.*

The $h(x) = ax + b$ attack discussed above is a special case of this theorem with $d = 1$. With classical queries finding a_0, \dots, a_d requires $d + 1$ queries, but with quantum queries the theorem shows that d queries are sufficient.

Theorem 1.2 is a quantum polynomial interpolation algorithm: given oracle access to the polynomial, the algorithm reconstructs its coefficients. This problem was studied previously by Kane and Kutin [KK11] who prove that $d/2$ quantum queries are insufficient to interpolate the polynomial. Interestingly, they conjecture that quantum interpolation requires $d + 1$ quantum queries as in the classical case, but Theorem 1.2 refutes that conjecture. Theorem 1.2 also applies to a quantum version of secret sharing where the shares themselves are superpositions. It shows that the classical Shamir secret sharing scheme [Sha79] is insecure if the shares are allowed to be quantum states obtained by evaluating the secret sharing polynomial on quantum superpositions.

More generally, the security of secret sharing schemes in the quantum settings was analyzed by Dam ard et al. [DFNS11].

As for one-time secure MACs, while pair-wise independence is insufficient for quantum one-time security, we show that four-wise independence is sufficient. That is, a four-way independent hash family gives rise to an existentially unforgeable MAC under a one-time quantum chosen message attack. It is still an open problem whether three-way independence is sufficient. More generally, we show that $(q + 1)$ -way independence is insufficient for a q -time quantum secure MAC, but $(3q + 1)$ -way independence is sufficient.

Motivation. Allowing the adversary to issue quantum chosen message queries is a natural and conservative security model and is therefore an interesting one to study. Showing that classical MAC constructions remain secure in this model gives confidence in case end-user computing devices eventually become quantum. Nevertheless, one might imagine that even in a future where computers are quantum, the last step in a MAC signing procedure is to sample the resulting quantum state so that the generated MAC is always classical. The quantum chosen message query model ensures that even if the attacker can bypass this last “classicalization” step, the MAC remains secure.

As further motivation we note that the results in this paper are the tip of a large emerging area of research with many open questions. Consider for example signature schemes. Can one design schemes that remain secure when the adversary can issue quantum chosen message queries? Similarly, can one design encryption systems that remain secure when the the adversary can issue quantum chosen ciphertext queries? More generally, for any cryptographic primitive modeled as an interactive game, one can ask how to design primitives that remain secure when the interaction between the adversary and its given oracles is quantum.

Other related work. Several recent works study the security of cryptographic primitives when the adversary can issue quantum queries [BDF⁺11, Zha12a, Zha12b]. So far these have focused on proving security of signatures, encryption, and identity-based encryption in the *quantum* random oracle model where the adversary can query the random oracle on superpositions of inputs. These works show that many, but not all, random oracle constructions remain secure in the quantum random oracle model. The quantum random oracle model has also been used to prove security of Merkle’s Puzzles in the quantum settings [BS08, BHK⁺11]. Meanwhile, Dam ard et al. [DFNS11] examine secret sharing and multiparty computation in a model where an adversary may corrupt a superposition of subsets of players, and build zero knowledge protocols that are secure, even when a dishonest verifier can issue challenges on superpositions.

Some progress toward identifying sufficient conditions under which classical protocols are also quantum immune has been made by Unruh [Unr10] and Hallgren et al. [HSS11]. Unruh shows that any scheme that is statistically secure in Cannetti’s universal composition (UC) framework [Can01] against classical adversaries is also statistically secure against quantum adversaries. Hallgren et al. show that for many schemes this is also true in the computational setting. These results, however, do not apply to MACs.

2 Preliminaries: Definitions and Notation

Let $[n]$ be the set $\{1, \dots, n\}$. For a prime power n , let \mathbb{F}_n be the finite field on n elements. For any positive integer n , let \mathbb{Z}_n be the ring of integers modulo n .

Functions will be denoted by capitol letters (such as F), and sets by capitol script letters (such as \mathcal{X}). We denote vectors with bold lower-case letters (such as \mathbf{v}), and the components of a vector $\mathbf{v} \in \mathcal{A}^n$ by v_i , $i \in [n]$. We denote matrices with bold capital letters (such as \mathbf{M}), and the components of a matrix $\mathbf{M} \in \mathcal{A}^{m \times n}$ by $M_{i,j}$, $i \in [m], j \in [n]$. Given a function $F : \mathcal{X} \rightarrow \mathcal{Y}$ and a vector $\mathbf{v} \in \mathcal{X}^n$, let $F(\mathbf{v})$ denote the vector $(F(v_1), F(v_2), \dots, F(v_k))$. Let $F([n])$ denote the vector $(F(1), F(2), \dots, F(n))$.

Given a vector space \mathcal{V} , let $\dim \mathcal{V}$ be the dimension of \mathcal{V} , or the number of vectors in any basis for \mathcal{V} . Given a set of vectors $\{\mathbf{v}_1, \dots, \mathbf{v}_k\}$, let $\text{span}\{\mathbf{v}_1, \dots, \mathbf{v}_k\}$ denote the space of all linear combinations of vectors in $\{\mathbf{v}_1, \dots, \mathbf{v}_k\}$. Given a subspace S of an inner-product space V , and a vector $\mathbf{v} \in V$, define $\text{proj}_S \mathbf{v}$ as the orthogonal projection of \mathbf{v} onto S , that is, the vector $\mathbf{w} \in S$ such that $\|\mathbf{v} - \mathbf{w}\|$ is minimized.

Given a matrix \mathbf{M} , we define the rank, denoted $\text{rank}(\mathbf{M})$, to be the size of the largest subset of rows (equivalently, columns) of \mathbf{M} that are linearly independent.

Given a function $F : \mathcal{X} \rightarrow \mathcal{Y}$ and a subset $\mathcal{S} \subseteq \mathcal{X}$, the restriction of F to \mathcal{S} is the function $F_{\mathcal{S}} : \mathcal{S} \rightarrow \mathcal{Y}$ where $F_{\mathcal{S}}(x) = F(x)$ for all $x \in \mathcal{S}$. A distribution D on the set of functions F from \mathcal{X} to \mathcal{Y} induces a distribution $D_{\mathcal{S}}$ on the set of functions from \mathcal{S} to \mathcal{Y} , where we sample from $D_{\mathcal{S}}$ by first sampling a function F from D , and outputting $F_{\mathcal{S}}$. We say that D is k -wise independent if, for each set \mathcal{S} of size at most k , each of the distributions $D_{\mathcal{S}}$ are truly random distributions on functions from \mathcal{S} to \mathcal{Y} . A set \mathcal{F} of functions from \mathcal{X} to \mathcal{Y} is k -wise independent if the uniform distribution on \mathcal{F} is k -wise independent.

2.1 Quantum Computation

The quantum system A is a complex Hilbert space \mathcal{H} with inner product $\langle \cdot | \cdot \rangle$. The state of a quantum system is given by a vector $|\psi\rangle$ of unit norm ($\langle \psi | \psi \rangle = 1$). Given quantum systems \mathcal{H}_1 and \mathcal{H}_2 , the joint quantum system is given by the tensor product $\mathcal{H}_1 \otimes \mathcal{H}_2$. Given $|\psi_1\rangle \in \mathcal{H}_1$ and $|\psi_2\rangle \in \mathcal{H}_2$, the product state is given by $|\psi_1\rangle |\psi_2\rangle \in \mathcal{H}_1 \otimes \mathcal{H}_2$. Given a quantum state $|\psi\rangle$ and an orthonormal basis $B = \{|b_0\rangle, \dots, |b_{d-1}\rangle\}$ for \mathcal{H} , a measurement of $|\psi\rangle$ in the basis B results in a value b_i with probability $|\langle b_i | \psi \rangle|^2$, and the state $|\psi\rangle$ is collapsed to the state $|b_i\rangle$. We let $b_i \leftarrow |\psi\rangle$ denote the distribution on b_i obtained by sampling $|\psi\rangle$.

A unitary transformation over a d -dimensional Hilbert space \mathcal{H} is a $d \times d$ matrix \mathbf{U} such that $\mathbf{U}\mathbf{U}^\dagger = \mathbf{I}_d$, where \mathbf{U}^\dagger represents the conjugate transpose. A quantum algorithm operates on a product space $\mathcal{H}_{in} \otimes \mathcal{H}_{out} \otimes \mathcal{H}_{work}$ and consists of n unitary transformations $\mathbf{U}_1, \dots, \mathbf{U}_n$ in this space. \mathcal{H}_{in} represents the input to the algorithm, \mathcal{H}_{out} the output, and \mathcal{H}_{work} the work space. A classical input x to the quantum algorithm is converted to the quantum state $|x, 0, 0\rangle$. Then, the unitary transformations are applied one-by-one, resulting in the final state

$$|\psi_x\rangle = \mathbf{U}_n \dots \mathbf{U}_1 |x, 0, 0\rangle.$$

The final state is measured, obtaining (a, b, c) with probability $|\langle a, b, c | \psi_x \rangle|^2$. The output of the algorithm is b .

Quantum-accessible Oracles. We will implement an oracle $O : \mathcal{X} \rightarrow \mathcal{Y}$ by a unitary transformation \mathbf{O} where

$$\mathbf{O}|x, y, z\rangle = |x, y + O(x), z\rangle$$

where $+: \mathcal{X} \times \mathcal{X} \rightarrow \mathcal{X}$ is some group operation on \mathcal{X} . Suppose we have a quantum algorithm that makes quantum queries to oracles O_1, \dots, O_q . Let $|\psi_0\rangle$ be the state of the algorithm before any queries, and let $\mathbf{U}_1, \dots, \mathbf{U}_q$ be the unitary transformations applied between queries. The final state of the algorithm will be

$$\mathbf{U}_q \mathbf{O}_q \dots \mathbf{U}_1 \mathbf{O}_1 |\psi_0\rangle$$

We can also have an algorithm make classical queries to O_i . In this case, the input to the oracle is measured before applying the transformation \mathbf{O}_i .

Fix an oracle $O: \mathcal{X} \rightarrow \mathcal{Y}$. Let $O^{(q)}: \mathcal{X}^q \rightarrow \mathcal{Y}^q$ be the oracle that maps \mathbf{x} into $O(\mathbf{x}) = (O(x_1), O(x_2), \dots, O(x_q))$. Observe that any quantum query to $O^{(q)}$ can be implemented using q quantum queries to O , where the unitary transformations between queries just permute the registers. We say that an algorithm that makes a single query to $O^{(q)}$ makes q non-adaptive queries to O .

The Density Matrix. Suppose the state of a quantum system depends on some hidden random variable $z \in \mathcal{Z}$, which is distributed according to a distribution D . That is, if the hidden variable is z , the state of the system is $|\psi_z\rangle$. We can then define the density matrix of the quantum system as

$$\rho = \sum_{z \in \mathcal{Z}} \Pr_D[z] |\psi_z\rangle \langle \psi_z|$$

Applying a unitary matrix \mathbf{U} to the quantum state corresponds to the transformation

$$\rho \rightarrow \mathbf{U} \rho \mathbf{U}^\dagger$$

A partial measurement on some registers has the effect of zeroing out the terms in ρ where those registers are not equal. For example, if we have two registers x and y , and we measure the x register, then the new density matrix is

$$\rho'_{x,y,x',y'} = \begin{cases} \rho_{x,y,x',y'} & \text{if } x = x' \\ 0 & \text{otherwise} \end{cases}$$

2.2 Quantum secure MACs

A MAC system comprises two algorithms: a (possibly) randomized MAC signing algorithm $S(k, m)$ and a MAC verification algorithm $V(k, m, t)$. Here k denotes the secret key chosen at random from the key space, m denotes a message in the message space, and t denotes the MAC tag in the tag space on the message m . These algorithms and spaces are parameterized by a security parameter λ .

Classically, a MAC system is said to be secure if no attacker can win the following game: a random key k is chosen from the key space and the attacker is presented with a signing oracle $S(k, \cdot)$. Queries to the signing oracle are called chosen message queries. Let $\{(m_i, t_i)\}_{i=1}^q$ be the set of message-tag pairs that the attacker obtains by interacting with the signing oracle. The attacker wins the game if it can produce an existential forgery, namely a valid message-tag pair (m^*, t^*) satisfying $(m^*, t^*) \notin \{(m_i, t_i)\}_{i=1}^q$. The MAC system is said to be secure if no “efficient” adversary can win this game with non-negligible probability in λ .

Quantum chosen message queries. In the quantum settings we allow the adversary to maintain its own quantum state and issue quantum queries to the signing oracle. Let $\sum_{m,x,y} \psi_{m,x,y} |m, x, y\rangle$ be the adversary’s state just prior to issuing a signing query. The MAC signing oracle transforms this state as follows:

1. it chooses a random string r that will be used by the MAC signing algorithm,
2. it signs each “slot” in the given superposition by running $S(k, m; r)$, that is running algorithm S with randomness r . More precisely, the signing oracle performs the following transformation:

$$\sum_{m,x,y} \psi_{m,x,y} |m, x, y\rangle \longrightarrow \sum_{m,x,y} \psi_{m,x,y} |m, x \oplus S(k, m; r), y\rangle$$

When the signing algorithm is deterministic there is no need to choose an r . However, for randomized signing algorithms the same randomness is used to compute the tag for all slots in the superposition. Alternatively, we could have required fresh randomness in every slot, but this would make it harder to implement the MAC system on a quantum device. Allowing the same randomness in every slot is more conservative and frees the signer from this concern. At any rate, the two models are very close — if need be, the random string r can be used as a key for a quantum-secure PRF [Zha12b] which is used to generate a fresh pseudorandom value for every slot.

Existential forgery. After issuing q quantum chosen message queries the adversary wins the game if it can generate $q + 1$ valid classical message-tag pairs.

Definition 2.1. *A MAC system is existentially unforgeable under a quantum chosen message attack (EUF-qCMA) if no adversary can win the quantum MAC game with non-negligible advantage in λ .*

Zhandry [Zha12b] gives an example of a classically secure PRF that is insecure under quantum queries. This PRF gives an example MAC that is classically secure, but insecure under quantum queries. Our goal for the remainder of the paper is to construct EUF-qCMA secure MACs.

3 The Rank Method

In this section we introduce the rank method which is a general approach to proving lower bounds on quantum algorithms. The setup is as follows: we give a quantum algorithm A access to some quantity $H \in \mathcal{H}$. By access, we mean that the final state of the algorithm is some fixed function of H . In this paper, \mathcal{H} will be a set of functions, and A will be given oracle access to $H \in \mathcal{H}$ by allowing A to make q quantum oracle queries to H , for some q . For now, we will treat \mathcal{H} abstractly, and return to the specific case where \mathcal{H} is a set of functions later.

The idea behind the rank method is that, if we treat the final states of the algorithm on different H as vectors, the space spanned by these vectors will be some subspace of the overall Hilbert space. If the dimension of this subspace is small enough, the subspace (and hence all of the vectors in it) must be reasonably far from most of the vectors in the measurement basis. This allows us to bound the ability of such an algorithm to achieve some goal.

For $H \in \mathcal{H}$, let $|\psi_H\rangle$ be the final state of the quantum algorithm A , before measurement, when given access to H . Suppose the different $|\psi_H\rangle$ vectors all lie in a space of dimension d . Let $\Psi_{A,\mathcal{H}}$ be the $|\mathcal{H}| \times d$ matrix whose rows are the various vectors $|\psi_H\rangle$.

Definition 3.1. For a quantum algorithm A given access to some value $H \in \mathcal{H}$, we define the rank, denoted $\text{rank}(A, \mathcal{H})$, as the rank of the matrix $\Psi_{A, \mathcal{H}}$.

The rank of an algorithm A seemingly contains very little information: it gives the dimension of the subspace spanned by the $|\psi_H\rangle$ vectors, but gives no indication of the orientation of this subspace or the positions of the $|\psi_H\rangle$ vectors in the subspace. Nonetheless, we demonstrate how the success probability of an algorithm can be bounded from above knowing only the rank of $\Psi_{A, \mathcal{H}}$.

Theorem 3.2. Let A be a quantum algorithm that has access to some value $H \in \mathcal{H}$ drawn from some distribution D and produces some output $w \in \mathcal{W}$. Let $R : \mathcal{H} \times \mathcal{W} \rightarrow \{\text{True}, \text{False}\}$ be a binary relation. Then the probability that A outputs some w such that $R(H, w) = \text{True}$ is at most

$$\left(\max_{w \in \mathcal{W}} \Pr_{H \leftarrow D} [R(H, w)] \right) \times \text{rank}(A, \mathcal{H}) .$$

In other words, the probability that A succeeds in producing $w \in \mathcal{W}$ for which $R(H, w)$ is true is at most $\text{rank}(A, \mathcal{H})$ times the best probability of success of any algorithm that ignores H and just outputs some fixed w .

Proof. The probability that A outputs a w such that $R(H, w) = \text{True}$ is

$$\Pr_{\substack{H \leftarrow D \\ w \leftarrow |\psi_H\rangle}} [R(H, w)] = \sum_H \Pr_D[H] \sum_{w: R(H, w)} |\langle w | \psi_H \rangle|^2 = \sum_w \sum_{H: R(H, w)} \Pr_D[H] |\langle w | \psi_H \rangle|^2$$

Now, $|\langle w | \psi_H \rangle|$ is just the magnitude of the projection of $|w\rangle$ onto the space spanned by the vector $|\psi_H\rangle$, that is, $\text{proj}_{\text{span}\{|\psi_H\rangle\}}(|w\rangle)$. This is at most the magnitude of the projection of $|w\rangle$ onto the space spanned by all of the $|\psi_{H'}\rangle$ for $H' \in \mathcal{H}$, or $\text{proj}_{\text{span}\{|\psi_{H'}\rangle\}}(|w\rangle)$. Thus,

$$\Pr_{\substack{H \leftarrow D \\ w \leftarrow |\psi_H\rangle}} [R(z, w)] \leq \sum_w \left(\sum_{H: R(H, w)} \Pr_D[H] \right) \left| \text{proj}_{\text{span}\{|\psi_{H'}\rangle\}}(|w\rangle) \right|^2$$

Now, we can perform the sum over H , which gives $\Pr_{H \leftarrow D} [R(H, w)]$. We can bound this by the maximum it attains over all w , giving us

$$\Pr_{\substack{H \leftarrow D \\ w \leftarrow |\psi_H\rangle}} [R(H, w)] \leq \left(\max_w \Pr_{H \leftarrow D} [R(H, w)] \right) \sum_w \left| \text{proj}_{\text{span}\{|\psi_{H'}\rangle\}}(|w\rangle) \right|^2$$

Now, let $|b_i\rangle$ be an orthonormal basis for $\text{span}\{|\psi_{H'}\rangle\}$. Then

$$\left| \text{proj}_{\text{span}\{|\psi_{H'}\rangle\}}(|w\rangle) \right|^2 = \sum_i |\langle b_i | w \rangle|^2$$

Summing over all w gives

$$\sum_w \left| \text{proj}_{\text{span}\{|\psi_{H'}\rangle\}}(|w\rangle) \right|^2 = \sum_w \sum_i |\langle b_i | w \rangle|^2 = \sum_i \sum_w |\langle b_i | w \rangle|^2$$

Since the w are the possible results of measurement, the vectors $|w\rangle$ form an orthonormal basis for the whole space, meaning $\sum_w |\langle b_i | w \rangle|^2 = |\langle b_i | \rangle|^2 = 1$. Hence, the sum just becomes the number of $|b_i\rangle$, which is just the dimension of the space spanned by the $|\psi_{H'}\rangle$. Thus,

$$\Pr_{\substack{H \leftarrow \mathcal{D} \\ w \leftarrow |\psi_H\rangle}} [R(H, w)] \leq \left(\max_{w \in \mathcal{W}} \Pr_{H \leftarrow \mathcal{D}} [R(H, w)] \right) (\dim \text{span}\{|\psi_{H'}\rangle\}) .$$

But $\dim \text{span}\{|\psi_{H'}\rangle\}$ is exactly $\text{rank}(\Psi_{A, \mathcal{H}}) = \text{rank}(A, \mathcal{H})$, which finishes the proof of the theorem. \square

We now move to the specific case of oracle access. \mathcal{H} is now some set of functions from \mathcal{X} to \mathcal{Y} , and our algorithm A makes q quantum oracle queries to a function $H \in \mathcal{H}$. Concretely, A is specified by $q + 1$ unitary matrices \mathbf{U}_i , and the final state of A on input H is the state

$$\mathbf{U}_q \mathbf{H} \mathbf{U}_{q-1} \cdots \mathbf{U}_1 \mathbf{H} \mathbf{U}_0 |0\rangle$$

where \mathbf{H} is the unitary transformation mapping $|x, y, z\rangle$ into $|x, y + H(x), z\rangle$, representing an oracle query to the function H . To use the rank method (Theorem 3.2) for our purposes, we need to bound the rank of such an algorithm. First, we define the following quantity:

$$C_{k,q,n} \equiv \sum_{r=0}^q \binom{k}{r} (n-1)^r .$$

Theorem 3.3. *Let \mathcal{X} and \mathcal{Y} be sets of size m and n and let H_0 be some function from \mathcal{X} to \mathcal{Y} . Let \mathcal{S} be a subset of \mathcal{X} of size k and let \mathcal{H} be some set of functions from \mathcal{X} to \mathcal{Y} that are equal to H_0 except possibly on points in \mathcal{S} . If A is a quantum algorithm making q queries to an oracle drawn from \mathcal{H} , then*

$$\text{rank}(A, \mathcal{H}) \leq C_{k,q,n} .$$

Proof. Let $|\psi_H^q\rangle$ be the final state of a quantum algorithm after q quantum oracle calls to an oracle $H \in \mathcal{H}$. We wish to bound the dimension of the space spanned by the vectors $|\psi_H^q\rangle$ for all $H \in \mathcal{H}$. We accomplish this by exhibiting a basis for this space. Our basis consists of $|\psi_{H'}^q\rangle$ vectors where H' only differs from H_0 at a maximum of q points in \mathcal{S} . We need to show that two things: that our basis consists of $C_{k,q,n}$ vectors, and that our basis does in fact span the whole space.

We first count the number of basis vectors by counting the number of H' oracles. For each r , there are $\binom{k}{r}$ ways of picking the subset \mathcal{T} of size r from \mathcal{S} where H' will differ from H_0 . For each subset \mathcal{T} , there are n^r possible functions H' . However, if any value $x \in \mathcal{T}$ satisfies $F(x) = H_0(x)$, then this is equivalent to a case where we remove x from \mathcal{T} , and we would have already counted this case for a smaller value of r . Thus, we can assume $H'(x) \neq H_0(x)$ for all x in \mathcal{T} . There are $(n-1)^r$ such functions. Summing over all r , we get that the number of distinct H' oracles is

$$\sum_{r=0}^q \binom{k}{r} (n-1)^r = C_{k,q,n} .$$

Next, we need to show that the $|\psi_{H'}^q\rangle$ vectors span the entire space of $|\psi_H^q\rangle$ vectors. We first introduce some notation: let $|\psi^0\rangle$ be the state of a quantum algorithm before any quantum queries. Let $|\psi_H^q\rangle$ be the state after q quantum oracle calls to the oracle H . Let

$$\mathbf{M}_H^q = \mathbf{U}_q \mathbf{H} \mathbf{U}_{q-1} \mathbf{H} \cdots \mathbf{U}_1 \mathbf{H} .$$

Then $|\psi_H^q\rangle = \mathbf{M}_H^q |\psi^0\rangle$.

We note that since $|\psi^0\rangle$ is fixed for any algorithm, it is sufficient to prove that the \mathbf{M}_H^q matrices are spanned by the $\mathbf{M}_{H'}^q$.

For any subset \mathcal{T} of \mathcal{S} , and a function $F : \mathcal{T} \rightarrow \mathcal{Y}$, let $J_{\mathcal{T},F}$ be the oracle such that

$$J_{\mathcal{T},F}(x) = \begin{cases} F(x) & \text{if } x \in \mathcal{T} \\ H_0(x) & \text{otherwise} \end{cases}.$$

Let $\mathbf{M}_{\mathcal{T},H}^q$ denote $\mathbf{M}_{J_{\mathcal{T},H}}^q$. In other words, $\mathbf{M}_{\mathcal{T},H}$ is the transformation matrix corresponding to the oracle that is equal to H on the set \mathcal{T} , and equal to H_0 elsewhere. We claim that any \mathbf{M}_H^q for $H \in \mathcal{H}_{\mathcal{S}}$ is a linear combination of the matrices $\mathbf{M}_{\mathcal{T},H}^q$ for subsets \mathcal{T} of \mathcal{S} of size at most q . We will fix a particular H , and for convenience of notation, we will let $J_{\mathcal{T}} = J_{\mathcal{T},H}$. That is, $J_{\mathcal{T}}$ is the oracle that is equal to H on the set \mathcal{T} and H_0 otherwise. We will also let $\mathbf{M}_{\mathcal{T}}^q = \mathbf{M}_{\mathcal{T},H}^q$ and $\mathbf{M}^q = \mathbf{M}_H^q$. That is, \mathbf{M}^q is the transition matrix corresponding to the oracle H , and $\mathbf{M}_{\mathcal{T}}$ is the transition matrix corresponding to using the oracle $J_{\mathcal{T}}$. For the singleton set $\{x\}$, we will also let $J_x = J_{\{x\}}$.

We make the following observations:

$$\mathbf{H} = \left(\sum_{x \in \mathcal{S}} \mathbf{J}_x \right) - (k-1)\mathbf{H}_0 \quad (3.1)$$

$$\mathbf{J}_{\mathcal{T}} = \left(\sum_{x \in \mathcal{T}} \mathbf{J}_x \right) - (|\mathcal{T}|-1)\mathbf{H}_0 \quad (3.2)$$

These identities can be seen by applying each side to the different inputs. Next, we take \mathbf{M}_H^q and $\mathbf{M}_{\mathcal{T}}^q$ and expand out the \mathbf{H} and $\mathbf{J}_{\mathcal{T}}$ terms using Equations 3.1 and 3.2:

$$\mathbf{M}^q = \mathbf{U}_q \left(\left(\sum_{x \in \mathcal{S}} \mathbf{J}_x \right) - (k-1)\mathbf{H}_0 \right) \mathbf{U}_{q-1} \cdots \mathbf{U}_1 \left(\left(\sum_{x \in \mathcal{S}} \mathbf{J}_x \right) - (k-1)\mathbf{H}_0 \right) \quad (3.3)$$

$$\mathbf{M}_{\mathcal{T}}^q = \mathbf{U}_q \left(\left(\sum_{x \in \mathcal{T}} \mathbf{J}_x \right) - (|\mathcal{T}|-1)\mathbf{H}_0 \right) \mathbf{U}_{q-1} \cdots \mathbf{U}_1 \left(\left(\sum_{x \in \mathcal{T}} \mathbf{J}_x \right) - (|\mathcal{T}|-1)\mathbf{H}_0 \right) \quad (3.4)$$

Let $\mathbf{J}_{\perp} = \mathbf{H}_0$. For a vector $\mathbf{r} \in (\mathcal{S} \cup \{\perp\})^q$, let

$$\mathbf{P}_{\mathbf{r}} = \mathbf{U}_q \mathbf{J}_{r_q} \mathbf{U}_{q-1} \cdots \mathbf{J}_{r_2} \mathbf{U}_1 \mathbf{J}_{r_1}$$

For a particular \mathbf{r} , we wish to expand the \mathbf{M}^q and $\mathbf{M}_{\mathcal{T}}^q$ matrices in terms of the $\mathbf{P}_{\mathbf{r}}$ matrices. If d of the components of \mathbf{r} are \perp , then the coefficient of $\mathbf{P}_{\mathbf{r}}$ in the expansion of \mathbf{M}^q is $(-1)^d (k-1)^d$. If, in addition, all of the other components of \mathbf{r} lie in \mathcal{T} , then the coefficient in the expansion of $\mathbf{M}_{\mathcal{T}}^q$ is $(-1)^d (|\mathcal{T}|-1)^d$ (if any of the components of \mathbf{r} lie outside of \mathcal{T} , the coefficient is 0).

Now, we claim that, for some values a_{ℓ} , we have

$$\mathbf{M}^q = \sum_{\ell=0}^q a_{\ell} \sum_{\mathcal{T} \subseteq \mathcal{S}: |\mathcal{T}|=\ell} \mathbf{M}_{\mathcal{T}}^q$$

To accomplish this, we look for the coefficient of $\mathbf{P}_{\mathbf{r}}$ in the expansion of the right hand side of this equation. Fix an ℓ . Let d be the number of components of \mathbf{r} equal to \perp , and let p be the

number of distinct component values other than \perp . Notice that $p + d \leq q$. Then there are $\binom{k-p}{\ell-p}$ different sets \mathcal{T} of size ℓ for which all of the values of the components lie in \mathcal{T} . Thus, the coefficient of $\mathbf{P}_{\mathbf{r}}$ is

$$\sum_{\ell=p}^q a_{\ell} \binom{k-p}{\ell-p} (-1)^i (\ell-1)^d$$

Therefore, we need values a_{ℓ} such that

$$\sum_{\ell=p}^q a_{\ell} \binom{k-p}{\ell-p} (\ell-1)^d = (k-1)^d \quad (3.5)$$

for all d, p . Notice that we can instead phrase this problem as a polynomial interpolation problem. The right hand side of Equation 3.5 is a polynomial P of degree $d \leq q - p$, evaluated at $k - 1$. We can interpolate this polynomial using the points $\ell = p, \dots, q$, obtaining

$$P(k-1) = \sum_{\ell=p}^q P(\ell-1) \prod_{j=p, j \neq \ell}^q \frac{k-p}{\ell-p}.$$

The numerator of the product evaluates to

$$\frac{(k-p)!}{(k-\ell)(k-q-1)!}$$

while to evaluate the bottom, we split it into two parts: $j = p, \dots, \ell - 1$ and $j = \ell + 1, \dots, q$. The first part evaluates to $(\ell - p)!$, and the second part evaluates to $(-1)^{q-\ell} (q - \ell)!$. With a little algebraic manipulation, we have that

$$P(k-1) = \sum_{\ell=p}^q P(\ell-1) \left(\binom{k-\ell-1}{k-q-1} (-1)^{q-\ell} \right) \binom{k-p}{\ell-p}$$

for all polynomials $P(x)$ of degree at most $q - p$. Setting $P(x) = x^d$ for $d = 0, \dots, q - \ell$, we see that Equation 3.5 is satisfied if

$$a_{\ell} = \binom{k-1-\ell}{k-1-q} (-1)^{q-\ell}.$$

□

3.1 An Example

Suppose our task is to, given one quantum query to an oracle $H : \mathcal{X} \rightarrow \mathcal{Y}$, produce two distinct pairs (x_0, y_0) and (x_1, y_1) such that $H(x_0) = y_0$ and $H(x_1) = y_1$. Suppose further that H is drawn from a pairwise independent set \mathcal{H} . We will now see that the rank method leads to a bound on the success probability of any quantum algorithm A .

Corollary 3.4. *No quantum algorithm A , making a single query to a function $H : \mathcal{X} \rightarrow \mathcal{Y}$ drawn from a pairwise independent set \mathcal{H} , can produce two distinct input/output pairs of H , except with probability at most $|\mathcal{X}|/|\mathcal{Y}|$.*

Proof. Let $m = |\mathcal{X}|$ and $n = |\mathcal{Y}|$. Since no outputs of H are fixed, we will set $\mathcal{S} = \mathcal{X}$ in Theorem 3.3, showing that the rank of the algorithm A is bounded by $C_{m,1,n} = 1 + m(n - 1) < mn$. If an algorithm makes no queries to H , the best it can do at outputting two distinct input/output pairs is to just pick two arbitrary distinct pairs, and output those. The probability that this zero-query algorithm succeeds is at most $1/n^2$. Then Theorem 3.2 tells us that A succeeds with probability at most $\text{rank}(A, \mathcal{H})$ times this amount, which equates to $\frac{m}{n}$. \square

For $m > n$, this bound is trivial. However, for m smaller than n , this gives a non-trivial bound, and for m exponentially smaller than n , this bound is negligible.

4 Outputting Values of a Random Oracle

In this section, we will prove Theorem 1.1. We consider the following problem: given q quantum queries to a random oracle $H : \mathcal{X} \rightarrow \mathcal{Y}$, produce $k > q$ distinct pairs (x_i, y_i) such that $y_i = H(x_i)$. Let $n = |\mathcal{Y}|$ be the size of the range. Motivated by our application to quantum-accessible MACs, we are interested in the case where the range \mathcal{Y} of the oracle is large, and we want to show that to produce even one extra input/output pair ($k = q + 1$) is impossible, except with negligible probability. We are also interested in the case where the range of the oracle, though large, is far smaller than the domain. Thus, the bound we obtained in the previous section (Corollary 3.4) is not sufficient for our purposes, since it is only non-trivial if the range is larger than the domain.

In the classical setting, when $k \leq q$, this problem is easy, since we can just pick an arbitrary set of k different x_i values, and query the oracle on each value. For $k > q$, no adversary of even unbounded complexity can solve this problem, except with probability $1/n^{k-q}$, since for any set of k inputs, at least $k - q$ of the corresponding outputs are completely unknown to the adversary. Therefore, for large n , we have a sharp threshold: for $k \leq q$, this problem can be solved efficiently with probability 1, and for even $k = q + 1$, this problem cannot be solved, even inefficiently, except with negligible probability.

In the quantum setting, the $k \leq q$ case is the same as before, since we can still query the oracle classically. However, for $k > q$, the quantum setting is more challenging. The adversary can potentially query the random oracle on a superposition of all inputs, so he “sees” the output of the oracle on all points. Proving that it is still impossible to produce k input/output pairs is thus more complicated, and existing methods fail to prove that this problem is difficult. Therefore, it is not immediately clear that we have the same sharp threshold as before.

In Section 4.1 we use the rank method to bound the probability that any (even computationally unbounded) quantum adversary succeeds. Then in Section 4.2 we show that our bound is tight by giving an efficient algorithm for this problem that achieves the lower bound. In particular, for an oracle $H : \mathcal{X} \rightarrow \mathcal{Y}$ we consider two cases:

- Exponentially-large range \mathcal{Y} and polynomial k, q . In this case, we will see that the success probability even when $k = q + 1$ is negligible. That is, to produce even one additional input/output pair is hard. Thus, we get the same sharp threshold as in the classical case
- Constant size range \mathcal{Y} and polynomial k, q . We show that even when q is a constant fraction of k we can still produce k input/output pairs with overwhelming probability using only q quantum queries. This is in contrast to the classical case, where the success probability for $q = ck$, $c < 1$, is negligible in k .

4.1 A Tight Upper Bound

Theorem 4.1. *Let A be a quantum algorithm making q queries to a random oracle $H : \mathcal{X} \rightarrow \mathcal{Y}$ whose range has size n , and produces $k > q$ pairs $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$. The probability that the x_i values are distinct and $y_i = H(x_i)$ for all $i \in [k]$ is at most $\frac{1}{n^k} C_{k,q,n}$.*

Proof. Before giving the complete proof, we sketch the special case where k is equal to the size of the domain. In this case, any quantum algorithm that outputs k distinct input/output pairs must output *all* input/output pairs. Similar to the proof of Corollary 3.4, we will set $\mathcal{S} = \mathcal{X}$, and use Theorem 3.3 to bound the rank of A at $C_{k,q,n}$. Now, any algorithm making *zero* queries succeeds with probability at most $1/n^k$. Theorem 3.2 then bounds the success probability of any q query algorithm as

$$\frac{1}{n^k} C_{k,q,n}.$$

Now for the general proof: first, we will assume that the probability A outputs any particular sequence of x_i values is independent of the oracle H . We will show how to remove this assumption later. We can thus write

$$|\psi_H^q\rangle = \sum_{\mathbf{x}} \alpha_{\mathbf{x}} |\mathbf{x}\rangle |\phi_{H,\mathbf{x}}\rangle$$

where $\alpha_{\mathbf{x}}$ are complex numbers whose square magnitudes sum to one, and $|\mathbf{x}\rangle |\phi_{H,\mathbf{x}}\rangle$ is the normalized projection of $|\psi_H^q\rangle$ onto the space spanned by $|\mathbf{x}, w\rangle$ for all w . The probability that A succeeds is equal to

$$\sum_H \Pr[H] \sum_{\mathbf{x}} |\langle \mathbf{x}, H(\mathbf{x}) | \psi_H^q \rangle|^2 = \sum_H \Pr[H] \sum_{\mathbf{x}} |\alpha_{\mathbf{x}}|^2 |\langle H(\mathbf{x}) | \phi_{H,\mathbf{x}} \rangle|^2.$$

First, we reorder the sums so the outer sum is the sum over \mathbf{x} . Now, we write $H = (H_0, H_1)$ where H_0 is the oracle restricted to the components of \mathbf{x} , and H_1 is the oracle restricted to all other inputs. Thus, our probability is:

$$\frac{1}{n^m} \sum_{\mathbf{x}} |\alpha_{\mathbf{x}}|^2 \sum_{H_0, H_1} \left| \langle H_0(\mathbf{x}) | \phi_{(H_0, H_1), \mathbf{x}} \rangle \right|^2.$$

Using the same trick as we did before, we can replace $|\langle H(\mathbf{x}) | \phi_{H,\mathbf{x}} \rangle|$ with the quantity

$$\left| \text{proj}_{\text{span}\{|\phi_{(H_0, H_1), \mathbf{x}}\rangle\}} |H_0(\mathcal{X})\rangle \right|,$$

which is bounded by $\left| \text{proj}_{\text{span}\{|\phi_{(H'_0, H_1), \mathbf{x}}\rangle\}} |H_0(\mathbf{x})\rangle \right|$ as we vary H'_0 over oracles whose domain is the components of \mathbf{x} . The probability of success is then bounded by

$$\frac{1}{n^m} \sum_{\mathbf{x}} |\alpha_{\mathbf{x}}|^2 \sum_{H_0, H_1} \left| \text{proj}_{\text{span}\{|\phi_{(H'_0, H_1), \mathbf{x}}\rangle\}} |H_0(\mathbf{x})\rangle \right|^2.$$

We now perform the sum over H_0 . Like in the proof of Corollary 3.4, the sum evaluates to $\dim \text{span}\{|\phi_{(H'_0, H_1), \mathbf{x}}\rangle\}$. Since the $|\phi_{(H'_0, H_1), \mathbf{x}}\rangle$ vectors are projections of $|\psi_H^q\rangle$, this dimension is bounded by $\dim \text{span}\{|\psi_{(H'_0, H_1)}^q\rangle\}$. Let \mathcal{H} be the set of oracles (H'_0, H_1) as we vary H'_0 , and consider A acting on oracles in \mathcal{H} . Fix some oracle H_0^* from among the H'_0 oracles, and let \mathcal{S} be the set of

components of \mathbf{x} . Then (H'_0, H_1) differs from (H_0^*, H_1) only on the elements of \mathcal{S} . Since $|\mathcal{S}| \leq k$, Theorem 3.2 tells us that $\text{rank}(A, \mathcal{H}) \leq C_{k,q,n}$. But

$$\text{rank}(A, \mathcal{H}) = \dim \text{span}\{|\psi_{(H'_0, H_1)}^q\rangle\}$$

Therefore, we can bound the success probability by

$$\frac{1}{n^m} \sum_{\mathbf{x}} |\alpha_{\mathbf{x}}|^2 \sum_{H_1} C_{k,q,n} .$$

Summing over all n^{m-k} different H_1 values and all \mathbf{x} values gives a bound of

$$\frac{1}{n^k} C_{k,q,n}$$

as desired.

So far, we have assume that A produces \mathbf{x} with probability independent of H . Now, suppose our algorithm A does not produce \mathbf{x} with probability independent of the oracle. We construct a new algorithm B with access to H that does the following: pick a random oracle O with the same domain and range as H , and give A the oracle $H + O$ that maps x into $H(x) + O(x)$. When A produces k input/output pairs (x_i, y_i) , output the pairs $(x_i, y_i - O(x_i))$. (x_i, y_i) are input/output pairs of $H + O$ if and only if $(x_i, y_i - O(x_i))$ are input/output pairs of H . Further, A still sees a random oracle, so it succeeds with the same probability as before. Moreover, the oracle A sees is now independent of H , so B outputs \mathbf{x} with probability independent of H . Thus, applying the above analysis to B shows that B , and hence A , produce k input/output pairs with probability at most

$$\frac{1}{n^k} C_{k,q,n}$$

□

For this paper, we are interested in the case where $n = |\mathcal{Y}|$ is exponentially large, and we are only allowed a polynomial number of queries. Suppose $k = q + 1$, the easiest non-trivial case for the adversary. Then, the probability of success is

$$\frac{1}{n^{q+1}} \sum_{r=0}^q \binom{q+1}{r} (n-1)^r = 1 - \left(1 - \frac{1}{n}\right)^{q+1} \leq \frac{q+1}{n} . \quad (4.1)$$

Therefore, to produce even one extra input/output pair is impossible, except with exponentially small probability, just like in the classical case. This proves the first part of Theorem 1.1.

4.2 The Optimal Attack

In this section, we present a quantum algorithm for the problem of computing $H(x_i)$ for k different x_i values, given only $q < k$ queries:

Theorem 4.2. *Let \mathcal{X} and \mathcal{Y} be sets, and fix integers $q < k$, and k distinct values $x_1, \dots, x_k \in \mathcal{X}$. There exists a quantum algorithm A that makes q non-adaptive quantum queries to any function $H : \mathcal{X} \rightarrow \mathcal{Y}$, and produces $H(x_1), \dots, H(x_k)$ with probability $C_{k,q,n}/n^k$, where $n = |\mathcal{Y}|$.*

The algorithm is similar to the algorithm of [vD98], though generalized to handle arbitrary range sizes. This algorithm has the same success probability as in Theorem 4.1, showing that both our attack and lower bound of Theorem 4.1 are optimal. This proves the second part of Theorem 1.1.

Proof. Assume that $\mathcal{Y} = \{0, \dots, n-1\}$. For a vector $\mathbf{y} \in \mathcal{Y}^k$, let $\Delta(\mathbf{y})$ be the number of coordinates of \mathbf{y} that do not equal 0. Also, assume that $x_i = i$.

Initially, prepare the state that is a uniform superposition of all vectors $\mathbf{y} \in \mathcal{Y}^k$ such that $\Delta(\mathbf{y}) \leq q$:

$$|\psi_1\rangle = \frac{1}{\sqrt{V}} \sum_{\mathbf{y}: \Delta(\mathbf{y}) \leq q} |\mathbf{y}\rangle$$

Notice that the number of vectors of length k with at most q non-zero coordinates is exactly

$$\sum_{r=0}^q \binom{k}{r} (n-1)^r = C_{k,q,n}.$$

We can prepare the state efficiently as follows: Let $\text{Setup}_{k,q,n} : [C_{k,q,n}] \rightarrow [n]^k$ be the following function: on input $\ell \in [C_{k,q,n}]$,

- Check if $\ell \leq C_{k-1,q,n}$. If so, compute the vector $\mathbf{y}' = \text{Setup}_{k-1,q,n}(n)$, and output the vector $\mathbf{y} = (0, \mathbf{y}')$.
- Otherwise, let $\ell' = \ell - C_{k-1,q,n}$. It is easy to verify that $\ell' \in [(n-1)C_{k-1,q-1,n}]$.
- Let $\ell'' \in C_{k-1,q-1,n}$ and $y_0 \in [n] \setminus \{0\}$ be the unique such integers such that $\ell' = (n-1)\ell'' + y_0 - n$.
- Let $\mathbf{y}' = \text{Setup}_{k-1,q-1,n}(\ell'')$, and output the vector $\mathbf{y} = (y_0, \mathbf{y}')$.

The algorithm relies on the observation that a vector \mathbf{y} of length k with at most q non-zero coordinates falls into one of either two categories:

- The first coordinate is 0, and the remaining $k-1$ coordinates form a vector with at most q non-zero coordinates
- The first coordinate is non-zero, and the remaining $k-1$ coordinates form a vector with at most $q-1$ non-zero coordinates.

There are $C_{k-1,q,n}$ vectors of the first type, and $C_{k-1,q-1,n}$ vectors of the second type for each possible setting of the first coordinate to something other than 0. Therefore, we divide $[A_{k,q,n}]$ into two parts: the first $C_{k-1,q,n}$ integers map to the first type, and the remaining $(n-1)C_{k-1,q-1,n}$ integers map to vectors of the second type.

We note that Setup is efficiently computable, invertible, and its inverse is also efficiently computable. Therefore, we can prepare $|\psi_1\rangle$ by first preparing the state

$$\frac{1}{\sqrt{C_{k,q,n}}} \sum_{\ell \in [C_{k,q,n}]} |\ell\rangle$$

and reversibly converting this state into $|\phi_1\rangle$ using $\text{Setup}_{k,q,n}$.

Next, let $F : \mathcal{Y}^k \rightarrow [k]^q$ be the function that outputs the indexes i such that $\mathbf{y}_i \neq 0$, in order of increasing i . If there are fewer than q such indexes, the function fills in the remaining spaces the

first indexes such that $\mathbf{y}_i = 0$. If there are more than q indexes, the function truncates to the first q . F is realizable by a simple classical algorithm, so it can be implemented as a quantum algorithm. Apply this algorithm to $|\psi_1\rangle$, obtaining the state

$$|\psi_2\rangle = \frac{1}{\sqrt{C_{k,q,n}}} \sum_{\mathbf{y}: \Delta(\mathbf{y}) \leq q} |\mathbf{y}, F(\mathbf{y})\rangle$$

Next, let $G : \mathcal{Y}^k \rightarrow \mathcal{Y}_q$ be the function that takes in vector \mathbf{y} , computes $\mathbf{x} = F(\mathbf{y})$, and outputs the vector $(y_{x_1}, y_{x_2}, \dots, y_{x_q})$. In other words, it outputs the vector of the non-zero components of \mathbf{y} , padding with zeros if needed. This function is also efficiently computable by a classical algorithm, so we can apply it to each part of the superposition:

$$|\psi_3\rangle = \frac{1}{\sqrt{C_{k,q,n}}} \sum_{\mathbf{y}: \Delta(\mathbf{y}) \leq q} |\mathbf{y}, F(\mathbf{y}), G(\mathbf{y})\rangle$$

Now we apply the Fourier transform to the $G(\mathbf{y})$ part, obtaining

$$|\psi_4\rangle = \frac{1}{\sqrt{C_{k,q,n}}} \sum_{\mathbf{y}: \Delta(\mathbf{y}) \leq q} |\mathbf{y}, F(\mathbf{y})\rangle \sum_{\mathbf{z}} e^{-i \frac{2\pi}{n} \langle \mathbf{z}, G(\mathbf{y}) \rangle} |\mathbf{z}\rangle$$

Now we can apply H to the $F(\mathbf{y})$ part using q non-adaptive queries, adding the answer to the \mathbf{z} part. The result is the state

$$|\psi_5\rangle = \frac{1}{\sqrt{C_{k,q,n}}} \sum_{\mathbf{y}: \Delta(\mathbf{y}) \leq q} |\mathbf{y}, F(\mathbf{y})\rangle \sum_{\mathbf{z}} e^{-i \frac{2\pi}{n} \langle \mathbf{z}, G(\mathbf{y}) \rangle} |\mathbf{z} + H(F(\mathbf{y}))\rangle$$

We can rewrite this last state as follows:

$$|\psi_5\rangle = \frac{1}{\sqrt{C_{k,q,n}}} \sum_{\mathbf{y}: \Delta(\mathbf{y}) \leq q} e^{i \frac{2\pi}{n} \langle H(F(\mathbf{y})), G(\mathbf{y}) \rangle} |\mathbf{y}, F(\mathbf{y})\rangle \sum_{\mathbf{z}} e^{-i \frac{2\pi}{n} \langle \mathbf{z}, G(\mathbf{y}) \rangle} |\mathbf{z}\rangle$$

Now, notice that $H(F(\mathbf{y}))$ is the vector of H applied to the indexes where \mathbf{y} is non-zero, and that $G(\mathbf{y})$ is the vector of values of \mathbf{y} at those points. Thus the inner product is

$$\langle H(F(\mathbf{y})), G(\mathbf{y}) \rangle = \sum_{i: y_i \neq 0} H(i) \times y_i = \sum_{i=0}^k H(i) y_i = \langle H([k]), \mathbf{y} \rangle.$$

The next step is to uncompute the \mathbf{z} and $F(\mathbf{y})$ registers, obtaining

$$|\psi_6\rangle = \frac{1}{\sqrt{C_{k,q,n}}} \sum_{\mathbf{y}: \Delta(\mathbf{y}) \leq q} e^{i \frac{2\pi}{n} \langle H([k]), \mathbf{y} \rangle} |\mathbf{y}\rangle$$

Lastly, we perform a Fourier transform the remaining space, obtaining

$$|\psi_7\rangle = \frac{1}{\sqrt{C_{k,q,n} n^k}} \sum_{\mathbf{z}} \left(\sum_{\mathbf{y}: \Delta(\mathbf{y}) \leq q} e^{i \frac{2\pi}{n} \langle H([k]) - \mathbf{z}, \mathbf{y} \rangle} \right) |\mathbf{z}\rangle$$

Now measure. The probability we obtain $H([k])$ is

$$\frac{1}{C_{k,q,n} n^k} \left| \sum_{\mathbf{y}: \Delta(\mathbf{y}) \leq q} 1 \right|^2 = \frac{C_{k,q,n}}{n^k}$$

as desired. □

As we have already seen, for exponentially-large \mathcal{Y} , this attack has negligible advantage for any $k > q$. However, if $n = |\mathcal{Y}|$ is constant, we can do better. The error probability is

$$\sum_{r=q+1}^k \binom{k}{r} \left(1 - \frac{1}{n}\right)^r \left(\frac{1}{n}\right)^{k-r} = \sum_{s=0}^{k-q-1} \binom{k}{s} \left(\frac{1}{n}\right)^s \left(1 - \frac{1}{n}\right)^{k-s}.$$

This is the probability that k consecutive coin flips, where each coin is heads with probability $1/n$, yields fewer than $k - q$ heads. Using the Chernoff bound, if $q > k(1 - 1/n)$, this probability is at most

$$e^{-\frac{n}{2k}(q-k(1-1/n))^2}.$$

For a constant n , let c be any constant with $1 - 1/n < c < 1$. If we use $q = ck$ queries, the error probability is less than

$$e^{-\frac{n}{2k}(k(c+1/n-1))^2} = e^{-\frac{nk}{2}(c+1/n-1)^2},$$

which is exponentially small in k . Thus, for constant n , and any constant c with $1 - 1/n < c < 1$, using $q = ck$ quantum queries, we can determine k input/output pairs with overwhelming probability. This is in contrast to the classical case, where with any constant fraction of k queries, we can only produce k input/output pairs with negligible probability. As an example, if H outputs two bits, it is possible to produce k input/output pairs of H using only $q = 0.8k$ quantum queries. However, with $0.8k$ classical queries, we can output k input/output pairs with probability at most $4^{-0.2k} < 0.76^k$.

5 Quantum-Accessible MACs

Using Theorem 4.1 we can now show that a quantum secure pseudorandom function [Zha12b] gives rise to the quantum-secure MAC, namely $S(k, m) = \text{PRF}(k, m)$. We prove that this mac is secure.

Theorem 5.1. *If $\text{PRF} : \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{Y}$ is a quantum-secure pseudorandom function and $1/|\mathcal{Y}|$ is negligible, then $S(k, m) = \text{PRF}(k, m)$ is a EUF-qCMA-secure MAC.*

Proof. Let A be a polynomial time adversary that makes q quantum queries to $S(k, \cdot)$ and produces $q + 1$ valid input/output pairs with probability ϵ . Let Game 0 be the standard quantum MAC attack game, where A makes q quantum queries to MAC_k . By definition, A 's success probability in this game is ϵ .

Let Game 1 be the same as Game 0, except that $S(k, \cdot)$ is replaced with a truly random function $O : \mathcal{X} \rightarrow \mathcal{Y}$, and define A 's success probability as the probability that A outputs $q + 1$ input/output pairs of O . Since PRF is a quantum-secure PRF, A 's advantage in distinguishing Game 0 from Game 1 is negligible.

Now, in Game 1, A makes q quantum queries to a random oracle, and tries to produce $q + 1$ input/output pairs. However, by Theorem 4.1 and Eq. (4.1) we know that A 's success probability is bounded by $(q + 1)/|\mathcal{Y}|$ which is negligible. It now follows that ϵ is negligible and therefore, S is a EUF-qCMA-secure MAC. \square

5.1 Carter-Wegman MACs

In this section, we show how to modify the Carter-Wegman MAC so that it is secure in the quantum setting presented in Section 2.2. Recall that H is an XOR-universal family of hash functions from \mathcal{X} into \mathcal{Y} if for any two distinct points x and y , and any constant $c \in \mathcal{Y}$,

$$\Pr_{h \leftarrow H}[H(x) - H(y) = c] = 1/|\mathcal{Y}|$$

The Carter-Wegman construction uses a pseudorandom function family PRF with domain \mathcal{X} and range \mathcal{Y} , and an XOR-universal family of hash functions \mathcal{H} from \mathcal{M} to \mathcal{Y} . The key is a pair (k, H) , where k is a key for PRF and H is a function drawn from \mathcal{H} . To sign a message, pick a random $r \in \mathcal{X}$, and return $(r, \text{PRF}(k, r) + H(m))$.

This MAC is not, in general, secure in the quantum setting presented in Section 2.2. The reason is that the same randomness is used in all slots of a quantum chosen message query, that is the signing oracle computes:

$$\sum_m \alpha_m |m\rangle \longrightarrow \sum_m \alpha_m |m, r, \text{PRF}(k, r) + H(m)\rangle$$

where the same r is used for all classical states of the superposition. For example, suppose \mathcal{H} is the set of functions $H(x) = ax + b$ for random a and b . With even a single quantum query, the adversary will be able to obtain a and $\text{PRF}(k, r) + b$ with high probability, using the algorithm from Theorem 6.2 in Section 6. Knowing both of these will allow the adversary to forge any message.

We show how to modify the standard Carter-Wegman MAC to make it secure in the quantum setting.

Construction 1 (Quantum Carter-Wegman). *The Quantum Carter-Wegman MAC (QCW-MAC) is built from a pseudorandom function PRF, an XOR-universal set of functions \mathcal{H} , and a pairwise independent set of functions \mathcal{R} .*

Keys: The secret key for QCW-MAC is a pair (k, H) , where k is a key for PRF and $H : \mathcal{M} \rightarrow \mathcal{Y}$ is drawn from \mathcal{H} .

Signing: To sign a message m choose a random $R \in \mathcal{R}$ and output the pair $(R(m), \text{PRF}(k, R(m)) + H(m))$ as the tag. When responding to a quantum chosen message query, the same R is used in all classical states of the superposition.

Verification: To verify that (r, s) is a valid tag for m , accept iff $\text{PRF}(k, r) + H(m) = s$.

Theorem 5.2. *The Quantum Carter-Wegman MAC is a EUF-qCMA secure MAC.*

Proof. We start with an adversary A that makes q tag queries, and then produces $q + 1$ valid message/tag pairs with probability ϵ . We now adapt the classical Carter-Wegman security proof to our MAC in the quantum setting.

When the adversary makes query i on the superposition

$$\sum_{m,y,z} \alpha_{m,y,z}^{(i)} |m, y, z\rangle,$$

the challenger responds with the superposition

$$\sum_{m,y,z} \alpha_{m,y,z}^{(i)} |m, y + S_i(m), z\rangle$$

where $S_i(m) = (R_i(m), \text{PRF}(k, (R_i(m)) + H(m)))$ for a randomly chosen $R_i \in \mathcal{R}$, where \mathcal{R} is a pairwise independent set of functions.

The adversary then creates $q+1$ triples (m_j, r_j, s_j) which, with probability ϵ , are valid message/tag tuples. That means $H(m_j) + \text{PRF}(k, r_j) = s_j$ for all j .

We now prove that ϵ must be small using a sequence of games:

Game 0: Run the standard MAC game, responding to query i with the oracle that maps m to $(R_i(m), \text{PRF}(k, R_i(m)) + H(m))$, where R_i is a random function from \mathcal{R} . The advantage of A in this game is the probability it produces $q+1$ forgeries. Denote this advantage as ϵ_0 , which is equal to ϵ .

Game 1: Replace $\text{PRF}(k, \cdot)$ with a truly random function F , and denote the advantage in this game as ϵ_1 . Since PRF is a quantum-secure PRF, ϵ_1 is negligibly close to ϵ_0 .

Game 2: Next we change the goal of the adversary. The adversary is now asked to produce a triple (m_0, m_1, s) where $H(m_0) - H(m_1) = s$. Given an adversary A for Game 1, we construct an adversary B for Game 2 as follows: run A , obtaining $q+1$ forgeries (m_j, r_j, s_j) such that $H(m_j) + F(r_j) = s_j$ with probability ϵ_1 . If all r_j are distinct, abort. Otherwise, assume without loss of generality that $r_0 = r_1$. Then

$$H(m_0) - H(m_1) = (s_0 - F(r_0)) - (s_1 - F(r_1)) = s_0 - s_1$$

so output $(m_0, m_1, s_0 - s_1)$. Let ϵ_2 be the advantage of B in this game. Let p be the probability that all r_j are distinct and A succeeds. Then $\epsilon_2 \geq \epsilon_1 - p$.

We wish to bound p . Define a new algorithm C , with oracle access to F , that first generates H , and then runs A , playing the role of challenger to A . When A outputs $q+1$ triples (m_j, r_j, s_j) , B outputs $q+1$ pairs $(r_j, s_j - H(m_j))$. If A succeeded, then $H(m_j) + F(r_j) = s_j$, so $F(r_j) = s_j - H(m_j)$, meaning the pairs C outputs are all input/output pairs of F . If all the r_j are distinct, then C will output $q+1$ input/output pairs, which is impossible except with probability at most $(q+1)/|\mathcal{Y}|$. Therefore, $p \leq (q+1)/|\mathcal{Y}|$. Therefore, as long as $|\mathcal{Y}|$ is super-polynomial in size, p is negligible, meaning ϵ_2 is negligibly close to ϵ_1 .

Game 3: Now modify the game so that we draw R_i uniformly at random from the set of all oracles. Notice that each R_i is queried only once, meaning pairwise-independent R_i look exactly like truly random R_i , so Game 3 looks exactly like Game 2 from the point of view of the adversary. Thus the success probability ϵ_3 is equal to ϵ_2 .

Game 4: For this game, we answer query i with the oracle that maps m to $(R_i(m), F(R_i(m)))$. That is, we ignore H for answering MAC queries. Let ϵ_4 be the success probability in this game.

To prove that ϵ_4 is negligibly close to ϵ_3 , we need the following lemma:

Lemma 5.3. *Consider two distributions D_1 and D_2 on oracles from \mathcal{M} into $\mathcal{X} \times \mathcal{Y}$:*

- D_1 : generate a random oracle $R : \mathcal{M} \rightarrow \mathcal{X}$ and a random oracle $P : \mathcal{M} \rightarrow \mathcal{Y}$, and output the oracle that maps m to $(R(m), P(m))$.
- D_2 : generate a random oracle $R : \mathcal{M} \rightarrow \mathcal{X}$ and a random oracle $F : \mathcal{X} \rightarrow \mathcal{Y}$, and output the oracle that maps m to $(R(m), F(R(m)))$.

Then the probability that any q -quantum query algorithm distinguishes D_1 from D_2 is at most $O(q^2/|\mathcal{X}|^{1/3})$.

Proof. Let B be a quantum algorithm making quantum queries that distinguishes with probability λ . We will now define a quantum algorithm C that is given r samples $(s_i, t_i) \in \mathcal{X} \times \mathcal{Y}$, where s_i are

chosen randomly, and t_i are either chosen randomly, or are equal to $T(s_i)$ for a randomly chosen function $T : \mathcal{X} \rightarrow \mathcal{Y}$. C 's goal is to distinguish these two cases. Notice that as long as the s_i are distinct, these two distributions are identical. Therefore, C 's distinguishing probability is at most the probability of a collision, which is at most $O(r^2/|\mathcal{X}|)$.

C works as follows: generate a random oracle $A : \mathcal{M} \rightarrow [r]$. Let $R(m) = s_{A(m)}$ and $P(m) = t_{A(m)}$, and give B the oracle $(R(m), P(m))$. If t_i are random, then we have the oracle that maps m to $(s_{A(m)}, t_{A(m)})$. This is exactly the small-range distribution of Zhandry [Zha12b], and is indistinguishable from D_1 except with probability $O(q^3/r)$.

Similarly, if $t_i = T(s_i)$, then the oracle maps m to $(s_{A(m)}, T(s_{A(m)}))$. The oracle that maps m to $s_{A(m)}$ is also a small-range distribution, so it is indistinguishable from a random oracle except with probability $O(q^3/r)$. If we replace $s_{A(m)}$ with a random oracle, we get exactly the distribution D_2 . Thus, D_2 is indistinguishable from $(s_{A(m)}, T(s_{A(m)}))$ except with probability $O(q^3/r)$.

Therefore, C 's success probability at distinguishing D_1 from D_2 is at least $\lambda - O(q^3/r)$, and is at most $O(r^2/|\mathcal{X}|)$. This means the distinguishing probability of B is at most

$$O\left(\frac{r^2}{|\mathcal{X}|} + \frac{q^3}{r}\right)$$

This is minimized by choosing $r = O(q|\mathcal{X}|^{1/3})$, which gives a distinguishing probability of at most $O(q^2/|\mathcal{X}|^{1/3})$. □

We show that ϵ_4 is negligibly-close to ϵ_3 using a sequence of sub-games. Game 3a is the game where we answer query i with the oracle that maps m to $(R_i(m), P_i(m) + H(m))$ where P_i is another random oracle. Notice that we can define oracles $R(i, m) = R_i(m)$ and $P(i, m) = P_i(m)$. Then R and P are random oracles, and using the above lemma, the success probability of B in Game 3a is negligibly close to that of Game 3. Notice that since P_i is random, $P'_i(m) = P_i(m) + H(m)$ is also random, so Game 3a is equivalent to the game where we answer query i with the oracle that maps m to $(R_i(m), P_i(m))$. Using the above lemma again, the success probability of B in this game is negligibly close to that of Game 4.

Now, we claim that ϵ_4 , the success probability in Game 4 is negligible. Indeed, the view of B is independent of H , so the probability that $H(m_0) - H(m_1) = s$ is $1/|\mathcal{Y}|$. Since ϵ_4 is negligibly close to $\epsilon = \epsilon_0$, the advantage of A , A 's advantage is also negligible. □

6 q-time MACs

In this section, we develop quantum one-time MACs, MACs that are secure when the adversary can issue only *one* quantum chosen message query. More generally, we will study quantum q -time MACs.

Classically, any pairwise independent function is a one-time MAC. In the quantum setting, Corollary 3.4 shows that when the range is much larger than the domain, this still holds. However, such MACs are not useful since we want the tag to be short. We first show that when the range is not larger than the domain, pairwise independence is not enough to ensure security:

Theorem 6.1. *For any set \mathcal{Y} of prime-power size, and any set \mathcal{X} with $|\mathcal{X}| \geq |\mathcal{Y}|$, there exist $(q+1)$ -wise independent functions from \mathcal{X} to \mathcal{Y} that are not q -time MACs.*

To prove this theorem, we treat \mathcal{Y} as a finite field, and assume $\mathcal{X} = \mathcal{Y}$, as our results are easy to generalize to larger domains. We use random degree q polynomials as our $(q + 1)$ -wise independent family, and show in Theorem 6.2 below that such polynomials can be completely recovered using only q quantum queries. It follows that the derived MAC cannot be q -time secure since once the adversary has the polynomial it can easily forge tags on new messages.

Theorem 6.2. *For any prime power n , there is an efficient quantum algorithm that makes only q quantum queries to an oracle implementing a degree- q polynomial $F : \mathbb{F}_n \rightarrow \mathbb{F}_n$, and completely determines F with probability $1 - O(qn^{-1})$.*

The theorem shows that a $(q + 1)$ -wise independence family is not necessarily a secure quantum q -time MAC since after q quantum chosen message queries the adversary extracts the entire secret key. The case $q = 1$ is particularly interesting. The following lemma will be used to prove Theorem 6.2:

Lemma 6.3. *For any prime power n , and any subset $\mathcal{X} \subseteq \mathbb{F}_n$ of size $n - k$, there is an efficient quantum algorithm that makes a single quantum query to any degree-1 polynomial $F : \mathcal{X} \rightarrow \mathbb{F}_n$, and completely determines F with probability $1 - O(kn^{-1})$.*

Proof. Write $F(x) = ax + b$ for values $a, b \in \mathbb{F}_n$, and write $n = p^t$ for some prime p and integer t . We design an algorithm to recover a and b .

Initialize the quantum registers to the state

$$|\psi_1\rangle = \frac{1}{\sqrt{n-k}} \sum_{x \in \mathcal{X}} |x, 0\rangle$$

Next, make a single oracle query to F , obtaining

$$|\psi_2\rangle = \frac{1}{\sqrt{n-k}} \sum_{x \in \mathcal{X}} |x, ax + b\rangle$$

Note that we can interpret elements $z \in \mathbb{F}_n$ as vectors $\mathbf{z} \in \mathbb{F}_p^t$. Let $\langle \mathbf{y}, \mathbf{z} \rangle$ be the inner product of vectors $\mathbf{y}, \mathbf{z} \in \mathbb{F}_p^t$. Multiplication by a in \mathbb{F}_n is a linear transformation over the vector space \mathbb{F}_p^t , and can therefore be represented by a matrix $\mathbf{M}_a \in \mathbb{F}_p^{t \times t}$. Thus, we can write

$$|\psi_2\rangle = \frac{1}{\sqrt{n-k}} \sum_{\mathbf{x} \in \mathcal{X}} |\mathbf{x}, \mathbf{M}_a \mathbf{x} + \mathbf{b}\rangle$$

Note that in the case $t = 1$, a is a scalar in \mathbb{F}_p , so \mathbf{M}_a is just the scalar a .

Now, the algorithm applies the Fourier transform to both registers, to obtain

$$|\psi_3\rangle = \frac{1}{n\sqrt{n-k}} \sum_{\mathbf{y}, \mathbf{z}} \left(\sum_{\mathbf{x} \in \mathcal{X}} \omega_p^{\langle \mathbf{x}, \mathbf{y} \rangle + \langle \mathbf{M}_a \mathbf{x} + \mathbf{b}, \mathbf{z} \rangle} \right) |\mathbf{y}, \mathbf{z}\rangle$$

where ω_p is a complex primitive p th root of unity.

The term in parenthesis can be written as

$$\left(\sum_{\mathbf{x} \in \mathcal{X}} \omega_p^{\langle \mathbf{x}, \mathbf{y} + \mathbf{M}_a^T \mathbf{z} \rangle} \right) \omega_p^{\langle \mathbf{b}, \mathbf{z} \rangle}$$

We will then do a change of variables, setting $\mathbf{y}' = \mathbf{y} + \mathbf{M}_a^T \mathbf{z}$.

Therefore, we can write the state as

$$|\psi_3\rangle = \frac{1}{n\sqrt{n-k}} \sum_{\mathbf{y}', \mathbf{z}} \left(\sum_{\mathbf{x} \in \mathcal{X}} \omega_p^{\langle \mathbf{x}, \mathbf{y}' \rangle} \right) \omega_p^{\langle \mathbf{b}, \mathbf{z} \rangle} |\mathbf{y}' - \mathbf{M}_a^T \mathbf{z}, \mathbf{z}\rangle$$

For $\mathbf{z} \neq 0$ and $\mathbf{y}' = 0$, we will now explain how to recover a from $(-\mathbf{M}_a^T \mathbf{z}, \mathbf{z})$. Notice that the transformation that takes \mathbf{a} and outputs $-\mathbf{M}_a^T \mathbf{z}$ is a linear transformation. Call this transformation \mathbf{L}_z . The coefficients of \mathbf{L}_z are easily computable, given \mathbf{z} , by applying the transformation to each of the unit vectors. Notice that if $t = 1$, \mathbf{L}_z is just the scalar $-z$. We claim that \mathbf{L}_z is invertible if $\mathbf{z} \neq 0$. Suppose there is some \mathbf{a} such that $\mathbf{L}_z \mathbf{a} = -\mathbf{M}_a^T \mathbf{z} = 0$. Since $\mathbf{z} \neq 0$, this means the linear operator $-\mathbf{M}_a^T$ is not invertible, so neither is $-\mathbf{M}_a$. But $-\mathbf{M}_a$ is just multiplication by $-a$ in the field \mathbb{F}_n . This multiplication is only non-invertible if $-a = 0$, meaning $\mathbf{a} = 0$, a contradiction. Therefore, the kernel of \mathbf{L}_z is just 0, so the map is invertible.

Therefore, to compute a , compute the inverse operator \mathbf{L}_z^{-1} and apply it to $-\mathbf{M}_a^T \mathbf{z}$, interpreting the result as a field element in \mathbb{F}_n . The result is a . More specifically, for $\mathbf{z} \neq 0$, apply the computation mapping (\mathbf{y}, \mathbf{z}) to $(\mathbf{L}_z^{-1} \mathbf{y}, \mathbf{z})$, which will take $(-\mathbf{M}_a^T \mathbf{z}, \mathbf{z})$ to (a, \mathbf{z}) . For $\mathbf{z} = 0$, we will just apply the identity map, leaving both registers as is. This map is now reversible, meaning this computation can be implemented as a quantum computation. The result is the state

$$|\psi_4\rangle = \frac{1}{n\sqrt{n-k}} \sum_{\mathbf{y}'} \left(\sum_{\mathbf{x} \in \mathcal{X}} \omega_p^{\langle \mathbf{x}, \mathbf{y}' \rangle} \right) \left(\sum_{\mathbf{z} \neq 0} \omega_p^{\langle \mathbf{b}, \mathbf{z} \rangle} |\mathbf{L}_z^{-1} \mathbf{y}' + a, \mathbf{z}\rangle + |\mathbf{y}', 0\rangle \right)$$

We will now get rid of the $|\mathbf{y}', 0\rangle$ terms by measuring whether $\mathbf{z} = 0$. The probability that $\mathbf{z} = 0$ is $1/n$, and in this case, we abort. Otherwise, we are left if the state

$$|\psi_5\rangle = \frac{1}{\sqrt{n(n-1)(n-k)}} \sum_{\mathbf{z} \neq 0, \mathbf{y}'} \left(\sum_{\mathbf{x} \in \mathcal{X}} \omega_p^{\langle \mathbf{x}, \mathbf{y}' \rangle} \right) \omega_p^{\langle \mathbf{b}, \mathbf{z} \rangle} |\mathbf{L}_z^{-1} \mathbf{y}' + a, \mathbf{z}\rangle$$

The algorithm then measures the first register. Recall that \mathcal{X} has size $n - k$. The probability the outcome of the measurement is a is then $(1 - k/n)$. In this case, we are left in the state

$$|\psi_6\rangle = \frac{1}{\sqrt{n-1}} \sum_{\mathbf{z} \neq 0} \omega_p^{\langle \mathbf{b}, \mathbf{z} \rangle} |\mathbf{z}\rangle$$

Next, the algorithm performs the inverse Fourier transform to the second register, arriving at the state

$$|\psi_7\rangle = \frac{1}{\sqrt{n(n-1)}} \sum_{\mathbf{w}} \left(\sum_{\mathbf{z} \neq 0} \omega_p^{\langle \mathbf{b} - \mathbf{w}, \mathbf{z} \rangle} \right) |\mathbf{w}\rangle$$

Now the algorithm measures again, and interpret the resulting vector as a field element. The probability that the result is b is $1 - 1/n$. Therefore, with probability $(1 - k/n)(1 - 1/n)^2 = 1 - O(k/n)$, the algorithm outputs both a and b .

□

Now we use this attack to obtain an attack on degree- d polynomials, for general d :

Proof of Theorem 6.2. We show how to recover the $q + 1$ different coefficients of any degree- q polynomial, using only $q - 1$ classical queries and a single quantum query.

Let a be the coefficient of x^q , and b the coefficient of x^{q-1} in $F(x)$. First, make $q - 1$ *classical* queries to arbitrary distinct points $\{x_1, \dots, x_{q-1}\}$. Let $Z(x)$ be the unique polynomial of degree $q - 2$ such that $r(x_i) = F(x_i)$, using standard interpolation techniques. Let $G(x) = F(x) - Z(x)$. $G(x)$ is a polynomial of degree q that is zero on the x_i , so it factors, allowing us to write

$$F(x) = Z(x) + (a'x + b') \prod_{i=1}^{q-1} (x - x_i)$$

By expanding the product, we see that $a = a'$ and $b = b' - a \sum x_i$. Therefore, we can implement an oracle mapping x to $a(x + \sum x_i) + b$ as follows:

- Query F on x , obtaining $F(x)$.
- Compute $Z(x)$, and let $G(x) = F(x) - Z(x)$.
- Output $G(x) / \prod (x - x_i) = a(x + \sum x_i) + b$.

This oracle works on all inputs except the $q - 1$ different x_i values. We run the algorithm from Lemma 6.3 on $\mathcal{X} = \mathbb{F}_n \setminus \{x_i\}$, we will recover with probability $1 - O(q/n)$ both a and $b + a \sum x_i$ using a single quantum query, from which we can compute a and b . Along with the $F(x_i)$ values, we can then reconstruct the entire polynomial. \square

6.1 Sufficient Conditions for a One-Time Mac

We show that, while pairwise independence is not enough for a one-time MAC, 4-wise independence is. We first generalize a theorem of Zhandry [Zha12a]:

Lemma 6.4. *Let A be any quantum algorithm that makes c classical queries and q quantum queries to an oracle H . If H is drawn from a $(c + 2q)$ -wise independent function, then the output distribution of A is identical to the case where H is truly random.*

Proof. If $q = 0$, then this theorem is trivial, since the c outputs A sees are distributed randomly. If $c = 0$, then the theorem reduces to that of Zhandry [Zha12a]. By adapting the proof of the $c = 0$ case to the general case, we get the lemma. Our approach is similar to the polynomial method, but needs to be adapted to handle classical queries correctly.

Our quantum algorithm makes $k = c + q$ queries. Let $Q \subseteq [k]$ be the set of queries that are quantum, and let $C \subseteq [k]$ be the set of queries that are classical.

Fix an oracle H . Let $\delta_{x,y}$ be 1 if $H(x) = y$ and 0 otherwise. Let $\rho^{(i)}$ be the density matrix after the i th query, and $\rho^{(i-1/2)}$ be the density matrix before the i th query. $\rho^{(q+1/2)}$ is the final state of the algorithm.

We now claim that $\rho^{(i)}$ and $\rho^{(i+1/2)}$ are polynomials of the $\delta_{x,y}$ of degree k_i , where k_i is twice the number of quantum queries made so far, plus the number of classical queries made so far.

$\rho^{(0)}$ and $\rho^{(0+1/2)}$ are independent of H , so they are not a function of the $\delta_{x,y}$ at all, meaning the degree is $0 = k_0$.

We now inductively assume our claim is true for $i - 1$, and express $\rho^{(i)}$ in terms of $\rho^{(i-1/2)}$. There are two cases:

- i is a quantum query. In this case, $k_i = k_{i-1} + 2$. We can write

$$\rho_{x,y,z,x',y',z'}^{(i)} = \rho_{x,y-H(x),z,x',y'-H(x'),z}^{(i-1/2)}$$

An alternative way to write this is as

$$\rho_{x,y,z,x',y',z'}^{(i)} = \sum_{r,r'} \delta_{x,y-r} \delta_{x',y'-r'} \rho_{x,r,z,x',r',z}^{(i-1/2)}$$

By induction, each of the $\rho_{x,r,z,x',r',z}^{(i-1/2)}$ are polynomials of degree k_{i-1} in the $d_{x,y}$ values, so $\rho_{x,y,z,x',y',z'}^{(i)}$ is a polynomial of degree $k_{i-1} + 2 = k_i$.

- i is a classical query. This means $l_i = k_{i-1} + 1$. Let $\rho^{(i-1/4)}$ representing the state after measuring the x register, but before making the actual query. This is identical to $\rho^{(i-1/2)}$, except the entries where $x \neq x'$ are zeroed out. We can then write

$$\rho_{x,y,z,x',y',z'}^{(i)} = \sum_{r,r'} \delta_{x,y-r} \delta_{x',y'-r'} \rho_{x,r,z,x',r',z}^{(i-1/4)} = \sum_{r,r'} \delta_{x,y-r} \delta_{x,y'-r'} \rho_{x,r,z,x,r',z}^{(i-1/2)}$$

Now, notice that $\delta_{x,y-r} \delta_{x,y'-r'}$ is zero unless $y - r = y' - r'$, in which case it just reduces to $\delta_{x,y-r}$. Therefore, we can simply further:

$$\rho_{x,y,z,x',y',z'}^{(i)} = \sum_r \delta_{x,y-r} \rho_{x,r,z,x,(y-y')+r,z}^{(i-1/2)}$$

By induction, each of the $\rho_{x,r,z,x,(y-y')+r,z}^{(i-1/2)}$ values are polynomials of degree k_{i-1} in the $d_{x,y}$ values, so $\rho_{x,y,z,x',y',z'}^{(i)}$ is a polynomial of degree $k_{i-1} + 1 = k_i$

Therefore, after all q queries, final matrix $\rho^{(q+1/2)}$ is a polynomial in the $d_{x,y}$ of degree at most $k = 2q + c$. We can then write the density matrix as

$$\rho^{(q+1/2)} = \sum_{\mathbf{x}, \mathbf{y}} \mathbf{M}_{\mathbf{x}, \mathbf{y}} \prod_{t=0}^{r_q} \delta_{x_t, y_t}$$

where \mathbf{x} and \mathbf{y} are vectors of length k , $\mathbf{M}_{\mathbf{x}, \mathbf{y}}$ are matrices, and the sum is over all possible vectors.

Now, fix a distribution D on oracles H . The density matrix for the final state of the algorithm, when the oracle is drawn from H , is given by

$$\sum_{\mathbf{x}, \mathbf{y}} \mathbf{M}_{\mathbf{x}, \mathbf{y}} \left(\sum_H \Pr[H \leftarrow D] \prod_{t=0}^{r_q} \delta_{x_t, y_t} \right)$$

The term in parenthesis evaluates to $\Pr_{H \leftarrow D}[H(\mathbf{x}) = \mathbf{y}]$. Therefore, the final density matrix can be expressed as

$$\sum_{\mathbf{x}, \mathbf{y}} \mathbf{M}_{\mathbf{x}, \mathbf{y}} \Pr_{H \leftarrow D}[H(\mathbf{x}) = \mathbf{y}]$$

Since \mathbf{x} and \mathbf{y} are vectors of length $k = 2q + c$, if D is k -wise independent, $\Pr_{H \leftarrow D}[H(\mathbf{x}) = \mathbf{y}]$ evaluates to the same quantity as if D was truly random. Thus the density matrices are the same. Since all of the statistical information about the final state of the algorithm is contained in the density matrix, the distributions of outputs are thus identical, completing the proof. \square

Using this lemma we show that $(3q + 1)$ -wise independence is sufficient for q -time MACs.

Theorem 6.5. *Any $(3q + 1)$ -wise independent family with domain \mathcal{X} and range \mathcal{Y} is a quantum q -time secure MAC provided $(q + 1)/|\mathcal{Y}|$ is negligible.*

Proof. Let D be some $(3q + 1)$ -wise independent function. Suppose we have an adversary A that makes q quantum queries to an oracle H , and attempts to produce $q + 1$ input/output pairs. Let ϵ_R be the probability of success when H is a random oracle, and let ϵ_D be the probability of success when H is drawn from D . We construct an algorithm B with access to H as follows: simulate A with oracle access to H . When A outputs $q + 1$ input/output pairs, simply make $q + 1$ queries to H to check that these are valid pairs. Output 1 if and only if all pairs are valid. Therefore, B makes q quantum queries and $c = q + 1$ classical queries to H , and outputs 1 if and only if A succeeds: if H is random, B outputs 1 with probability ϵ_R , and if H is drawn from D , B outputs 1 with probability ϵ_D . Now, since D is $(3q + 1)$ -wise independent and $3q + 1 = 2q + c$, Lemma 6.4 shows that the distributions of outputs when H is drawn from D is identical to that when H is random, meaning $\epsilon_D = \epsilon_R$.

Thus, when H is drawn from D , A 's succeeds with the same probability that it would if H was random. But we already know that if H is truly random, A 's success probability is less than $(q + 1)/|\mathcal{Y}|$. Therefore, when H is drawn from D , A succeeds with probability less than $(q + 1)/|\mathcal{Y}|$, which is negligible. Hence, if H is drawn from D , H is a q -time MAC. \square \square

7 Conclusion

We introduced the rank method as a general technique for obtaining lower bounds on quantum oracle algorithms and used this method to bound the probability that a quantum algorithm can evaluate a random oracle $\mathcal{O} : \mathcal{X} \rightarrow \mathcal{Y}$ at k points using $q < k$ queries. When the range of \mathcal{Y} is small, say $|\mathcal{Y}| = 8$, a quantum algorithm can recover k points of \mathcal{O} from only $0.9k$ queries with high probability. However, we show that when the range \mathcal{Y} is large, no algorithm can produce k input-output pairs of \mathcal{O} using only $k - 1$ queries, with non-negligible probability. We use these bounds to construct the first MACs secure against quantum chosen message attacks. We consider both PRF and Carter-Wegman constructions. For one-time MACs we showed that pair-wise independence does not ensure security, but four-way independence does.

These results suggest many directions for future work. First, can these bounds be generalized to signatures to obtain signatures secure against quantum chosen message attacks? Similarly, can we construct encryption systems secure against quantum chosen ciphertext attacks where decryption queries are superpositions of ciphertexts?

Acknowledgments

We thank Luca Trevisan and Amit Sahai for helpful conversations about this work. This work was supported by NSF, DARPA, IARPA, the Air Force Office of Scientific Research (AFO SR) under a MURI award, Samsung, and a Google Faculty Research Award. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA, IARPA, DoI/NBC, or the U.S. Government.

References

- [Aar02] Scott Aaronson. Quantum lower bound for the collision problem. In *STOC*, pages 635–642, 2002.
- [Amb00] Andris Ambainis. Quantum lower bounds by quantum arguments. In *STOC*, pages 636–643, 2000.
- [Amb06] Andris Ambainis. Polynomial degree vs. quantum query complexity. *J. Comput. Syst. Sci.*, 72(2):220–238, 2006.
- [ASdW09] Andris Ambainis, Robert Spalek, and Ronald de Wolf. A new quantum lower bound method, with applications to direct product theorems and time-space tradeoffs. *Algorithmica*, 55(3):422–461, 2009.
- [BBC⁺01] Robert Beals, Harry Buhrman, Richard Cleve, Michele Mosca, and Ronald de Wolf. Quantum Lower Bounds by Polynomials. *Journal of the ACM (JACM)*, 48(4):778–797, July 2001.
- [BCK96] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Pseudorandom functions revisited: The cascade construction and its concrete security. In *FOCS*, pages 514–523, 1996.
- [BDF⁺11] Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. Random Oracles in a Quantum World. In *Advances in Cryptology — ASIACRYPT 2011*, 2011.
- [BHK⁺11] Gilles Brassard, Peter Høyer, Kassem Kalach, Marc Kaplan, Sophie Laplante, and Louis Salvail. Merkle Puzzles in a Quantum World. *Advances in Cryptology - CRYPTO 2011*, pages 391–410, 2011.
- [BKR00] Mihir Bellare, Joe Kilian, and Phillip Rogaway. The security of the cipher block chaining message authentication code. *J. Comput. Syst. Sci.*, 61(3), 2000.
- [BS08] Gilles Brassard and Louis Salvail. Quantum Merkle Puzzles. *Second International Conference on Quantum, Nano and Micro Technologies (ICQNM 2008)*, pages 76–79, February 2008.
- [Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proc. of FOCS*. IEEE, 2001.
- [DFNS11] Ivan Damgård, Jakob Funder, Jesper Buus Nielsen, and Louis Salvail. Superposition attacks on cryptographic protocols. *CoRR*, abs/1108.6313, 2011.
- [GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to Construct Random Functions. *Journal of the ACM (JACM)*, 33(4):792–807, 1986.
- [HSS11] Sean Hallgren, Adam Smith, and Fang Song. Classical cryptographic protocols in a quantum world. In *Proc. of Crypto*, LNCS. Springer, 2011.
- [KdW03] Iordanis Kerenidis and Ronald de Wolf. Exponential lower bound for 2-query locally decodable codes via a quantum argument. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC)*, pages 106–115, 2003.

- [KK11] Daniel M. Kane and Samuel A. Kutin. Quantum interpolation of polynomials. *Quantum Information & Computation*, 11(1&2):95–103, 2011. First published in 2009.
- [Sha79] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.
- [Unr10] Dominique Unruh. Universally Composable Quantum Multi-Party Computation. *Advances in Cryptology — EUROCRYPT 2010*, pages 486–505, 2010.
- [vD98] Wim van Dam. Quantum oracle interrogation: Getting all information for almost half the price. In *FOCS*, pages 362–367, 1998.
- [WC81] Mark N. Wegman and Larry Carter. New hash functions and their use in authentication and set equality. *J. Comput. Syst. Sci.*, 22(3):265–279, 1981.
- [Zha12a] Mark Zhandry. Secure Identity-Based Encryption in the Quantum Random Oracle Model. In *Advances in Cryptology — CRYPTO*, 2012. Full version available at the Cryptology ePrint Archives: <http://eprint.iacr.org/2012/076/>.
- [Zha12b] Mark Zhandry. How to Construct Quantum Random Functions. In *Proceedings of FOCS*, 2012. Full version available at the Cryptology ePrint Archives: <http://eprint.iacr.org/2012/182/>.

Dynamic Proofs of Retrievability via Oblivious RAM

David Cash*

Alptekin Küpçü†

Daniel Wichs‡

March 15, 2013

Abstract

Proofs of retrievability allow a client to store her data on a remote server (e.g., “in the cloud”) and periodically execute an efficient *audit* protocol to check that all of the data is being maintained correctly and can be recovered from the server. For efficiency, the *computation* and *communication* of the server and client during an audit protocol should be significantly smaller than reading/transmitting the data in its entirety. Although the server is only asked to access a few locations of its storage during an audit, it must *maintain full knowledge of all client data* to be able to pass.

Starting with the work of Juels and Kaliski (CCS ’07), all prior solutions to this problem crucially assume that the client data is *static* and do not allow it to be efficiently updated. Indeed, they all store a redundant encoding of the data on the server, so that the server must delete a large fraction of its storage to ‘lose’ any actual content. Unfortunately, this means that even a single bit modification to the original data will need to modify a large fraction of the server storage, which makes updates highly inefficient. Overcoming this limitation was left as the main open problem by all prior works.

In this work, we give the first solution providing proofs of retrievability for *dynamic* storage, where the client can perform arbitrary reads/writes on any location within her data by running an efficient protocol with the server. At any point in time, the client can execute an efficient audit protocol to ensure that the server maintains the *latest version* of the client data. The computation and communication complexity of the server and client in our protocols is only *polylogarithmic* in the size of the client’s data. The starting point of our solution is to split up the data into small blocks and redundantly encode each block of data individually, so that an update inside any data block only affects a few codeword symbols. The main difficulty is to prevent the server from identifying and deleting too many codeword symbols belonging to any single data block. We do so by hiding where the various codeword symbols for any individual data block are stored on the server and when they are being accessed by the client, using the algorithmic techniques of *oblivious RAM*.

*IBM Research, T.J. Watson. Hawthorne, NY, USA. cdc@gatech.edu

†Koç University. İstanbul, TURKEY. akupcu@ku.edu.tr

‡IBM Research, T.J. Watson. Hawthorne, NY, USA. wichs@cs.nyu.edu

1 Introduction

Cloud storage systems (Amazon S3, Dropbox, Google Drive etc.) are becoming increasingly popular as a means of storing data reliably and making it easily accessible from any location. Unfortunately, even though the remote storage provider may not be trusted, current systems provide few security or integrity guarantees.

Guaranteeing the *privacy* and *authenticity* of remotely stored data while allowing efficient access and updates is non-trivial, and relates to the study of *oblivious RAMs* and *memory checking*, which we will return to later. The main focus of this work, however, is an orthogonal question: How can we efficiently verify that the entire client data is being stored on the remote server in the first place? In other words, what prevents the server from deleting some portion of the data (say, an infrequently accessed sector) to save on storage?

Provable Storage. Motivated by the questions above, there has been much cryptography and security research in creating a provable storage mechanism, where an untrusted server can *prove* to a client that her data is kept intact. More precisely, the client can run an efficient *audit* protocol with the untrusted server, guaranteeing that the server can only pass the audit if it maintains full *knowledge* of the entire client data. This is formalized by requiring that the data can be efficiently *extracted* from the server given its state at the beginning of any successful audit. One may think of this as analogous to the notion of extractors in the definition of *zero-knowledge proofs of knowledge* [14, 4].

One trivial audit mechanism, which accomplishes the above, is for the client to simply download all of her data from the server and check its authenticity (e.g., using a MAC). However, for the sake of efficiency, we insist that the *computation* and *communication* of the server and client during an audit protocol is much smaller than the potentially huge size of the client’s data. In particular, the server shouldn’t even have to *read* all of the client’s data to run the audit protocol, let alone *transmit* it. A scheme that accomplishes the above is called a *Proof of Retrievability* (PoR).

Prior Techniques. The first PoR schemes were defined and constructed by Juels and Kaliski [19], and have since received much attention. We review the prior work and closely related primitives (e.g., *sublinear authenticators* [23] and *provable data possession* [1]) in Section 1.2.

On a very high level, all PoR constructions share essentially the same common structure. The client stores some *redundant encoding* of her data under an erasure code on the server, ensuring that the server must delete a significant fraction of the encoding before losing any actual data. During an audit, the client then checks a few random locations of the encoding, so that a server who deleted a significant fraction will get caught with overwhelming probability.

More precisely, let us model the client’s input data as a string $\mathbf{M} \in \Sigma^\ell$ consisting of ℓ symbols from some small alphabet Σ , and let $\text{Enc} : \Sigma^\ell \rightarrow \Sigma^{\ell'}$ denote an erasure code that can correct the erasure of up to $\frac{1}{2}$ of its output symbols. The client stores $\text{Enc}(\mathbf{M})$ on the server. During an audit, the client selects a small random subset of t out of the ℓ' locations in the encoding, and challenges the server to respond with the corresponding values, which it then checks for authenticity (e.g., using MAC tags). Intuitively, if the server deletes more than half of the values in the encoding, it will get caught with overwhelming probability $> 1 - 2^{-t}$ during the audit, and otherwise it retains knowledge of the original data because of the redundancy of the encoding. The complexity of the audit protocol is only proportional to t which can be set to the *security parameter* and is independent of the size of the client data.¹

Difficulty of Updates. One of the main limitations of all prior PoR schemes is that they do not support efficient updates to the client data. Under the above template for PoR, if the client wants to modify even a single location of \mathbf{M} , it will end up needing to change the values of at least half of the locations in $\text{Enc}(\mathbf{M})$

¹Some of the more advanced PoR schemes (e.g., [27, 10]) optimize the communication complexity of the audit even further by cleverly compressing the t codeword symbols and their authentication tags in the server’s response.

on the server, requiring a large amount of work (linear in the size of the client data). Constructing a PoR scheme that allows for efficient updates was stated as the main open problem by Juels and Kaliski [19]. We emphasize that, in the setting of updates, the audit protocol must ensure that the server correctly maintains knowledge of the *latest version* of the client data, which includes all of the changes incurred over time. Before we describe our solution to this problem, let us build some intuition about the challenges involved by examining two natural but *flawed* proposals.

First Proposal. A natural attempt to overcome the inefficiency of updating a huge redundant encoding is to encode the data “locally” so that a change to one position of the data only affects a small number of codeword symbols. More precisely, instead of using an erasure code that takes all ℓ data symbols as input, we can use a code $\text{Enc} : \Sigma^k \rightarrow \Sigma^n$ that works on small blocks of only $k \ll \ell$ symbols encoded into n symbols. The client divides the data \mathbf{M} into $L = \ell/k$ *message blocks* $(\mathbf{m}_1, \dots, \mathbf{m}_L)$, where each block $\mathbf{m}_i \in \Sigma^k$ consists of k symbols. The client redundantly encodes each message block \mathbf{m}_i individually into a corresponding *codeword block* $\mathbf{c}_i = \text{Enc}(\mathbf{m}_i) \in \Sigma^n$ using the above code with small inputs. Finally the client concatenates these codeword blocks to form the value $\mathbf{C} = (\mathbf{c}_1, \dots, \mathbf{c}_L) \in \Sigma^{Ln}$, which it stores on the server. Auditing works as before: The client randomly chooses t of the $L \cdot n$ locations in \mathbf{C} and challenges the server to respond with the corresponding codeword symbols in these locations, which it then tests for authenticity.² The client can now read/write to any location within her data by simply reading/writing to the n relevant codeword symbols on the server.

The above proposal can be made secure when the block-size k (which determines the complexity of reads/updates) and the number of challenged locations t (which determines the complexity of the audit) are both set to $\Omega(\sqrt{\ell})$ where ℓ is the size of the data (see Appendix A for details). This way, the audit is likely to check sufficiently many values in *each* codeword block \mathbf{c}_i . Unfortunately, if we want a truly efficient scheme and set $n, t = o(\sqrt{\ell})$ to be small, then this solution becomes completely insecure. The server can delete a single codeword block \mathbf{c}_i from \mathbf{C} entirely, losing the corresponding message block \mathbf{m}_i , but still maintain a good chance of passing the above audit as long as none of the t random challenge locations coincides with the n deleted symbols, which happens with good probability.

Second Proposal. The first proposal (with small n, t) was insecure because a cheating server could easily identify the locations within \mathbf{C} that correspond to a single message block and delete exactly the codeword symbols in these locations. We can prevent such attacks by pseudo-randomly permuting the locations of all of the different codeword-symbols of different codeword blocks together. That is, the client starts with the value $\mathbf{C} = (\mathbf{C}[1], \dots, \mathbf{C}[Ln]) = (\mathbf{c}_1, \dots, \mathbf{c}_L) \in \Sigma^{Ln}$ computed as in the first proposal. It chooses a pseudo-random permutation $\pi : [Ln] \rightarrow [Ln]$ and computes the permuted value $\mathbf{C}' := (\mathbf{C}[\pi(1)], \dots, \mathbf{C}[\pi(Ln)])$ which it then stores on the server in an encrypted form (each codeword symbol is encrypted separately). The audit still checks t out of Ln random locations of the server storage and verifies authenticity.

It may seem that the server now cannot immediately identify and *selectively* delete codeword-symbols belonging to a single codeword block, thwarting the attack on the first proposal. Unfortunately, this modification only re-gains security in the static setting, when the client never performs any operations on the data.³ Once the client wants to update some location of \mathbf{M} that falls inside some message block \mathbf{m}_i , she has to reveal to the server where all of the n codeword symbols corresponding to $\mathbf{c}_i = \text{Enc}(\mathbf{m}_i)$ reside in its storage since she needs to update exactly these values. Therefore, the server can later selectively delete exactly these n codeword symbols, leading to the same attack as in the first proposal.

Impossibility? Given the above failed attempts, it may even seem that truly efficient updates could be inherently incompatible with efficient audits in PoR. If an update is efficient and only changes a small

²This requires that we can efficiently check the *authenticity* of the remotely stored data \mathbf{C} , while supporting efficient updates on it. This problem is solved by *memory checking* (see our survey of related work in Section 1.2).

³A variant of this idea was actually used by Juels and Kaliski [19] for extra efficiency in the static setting.

subset of the server’s storage, then the server can always just *ignore* the update, thereby failing to maintain knowledge of the latest version of the client data. All of the prior techniques appear ineffective against such attack. More generally, any audit protocol which just checks a *small subset of random* locations of the server’s storage is unlikely to hit any of the locations involved in the update, and hence will not detect such cheating, meaning that it cannot be secure.⁴ However, this does not rule out the possibility of a very efficient solution that relies on a more clever audit protocol, which is likelier to check recently updated areas of the server’s storage and therefore detect such an attack. Indeed, this property will be an important component in our actual solution.

1.1 Our Results and Techniques

Overview of Result. In this work, we give the first solution to *dynamic PoR* that allows for efficient updates to client data. The client only keeps some short local state, and can execute arbitrary read/write operations on any location within the data by running a corresponding protocol with the server. At any point in time, the client can also initiate an audit protocol, which ensures that a passing server must have complete knowledge of the *latest version* of the client data. The cost of any read/write/audit execution in terms of server/client work and communication is only *polylogarithmic* in the size of the client data. The server’s storage remains linear in the size of the client data. Therefore, our scheme is optimal in an asymptotic sense, up to polylogarithmic factors. See Section 7 for a detailed efficiency analysis.

PoR via Oblivious RAM. Our dynamic PoR solution starts with the same idea as the first proposal above, where the client redundantly encodes small blocks of her data individually to form the value $\mathbf{C} = (\mathbf{c}_1, \dots, \mathbf{c}_L) \in \Sigma^{Ln}$, consisting of L codeword blocks and $\ell' = Ln$ codeword symbols, as defined previously. The goal is to then store \mathbf{C} on the server in some “clever way” so that the server cannot selectively delete too many symbols within any single codeword block \mathbf{c}_i , even after observing the client’s read and write executions (which access exactly these symbols). As highlighted by the second proposal, simply permuting the locations of the codeword symbols of \mathbf{C} is insufficient. Instead, our main idea is to store all of the individual codeword symbols of \mathbf{C} on the server using an *oblivious RAM* scheme.

Overview of ORAM. Oblivious RAM (ORAM), initially defined by Goldreich and Ostrovsky [13], allows a client to outsource her *memory* to a remote server while allowing the client to perform random-access reads and writes in a *private* way. More precisely, the client has some data $\mathbf{D} \in \Sigma^d$, which she stores on the server in some carefully designed privacy-preserving form, while only keeping a short local state. She can later run efficient protocols with the server to read or write to the individual entries of \mathbf{D} . The read/write protocols of the ORAM scheme should be efficient, and the client/server work and communication during each such protocol should be small compared to the size of \mathbf{D} (e.g., *polylogarithmic*). A secure ORAM scheme not only hides the *content* of \mathbf{D} from the server, but also the *access pattern* of which *locations* in \mathbf{D} the client is reading or writing in each protocol execution. Thus, the server cannot discern any correlation between the physical locations of its storage that it is asked to access during each read/write protocol execution and the logical location inside \mathbf{D} that the client wants to access via this protocol.

We review the literature and efficiency of ORAM schemes in Section 6. In our work, we will also always use ORAM schemes that are *authenticated*, which means that the client can detect if the server ever sends an incorrect value. In particular, authenticated ORAM schemes ensure that the most recent version of the data is being retrieved in any accepting read execution, preventing the server from “rolling back” updates.

Construction of Dynamic PoR. A detailed technical description of our construction appears in Section 5, and below we give a simplified overview. In our PoR construction, the client starts with data

⁴The above only holds when the complexity of the updates and the audit are both $o(\sqrt{\ell})$, where ℓ is the size of the data. See Appendix A for a simple protocol of this form that archives square-root complexity.

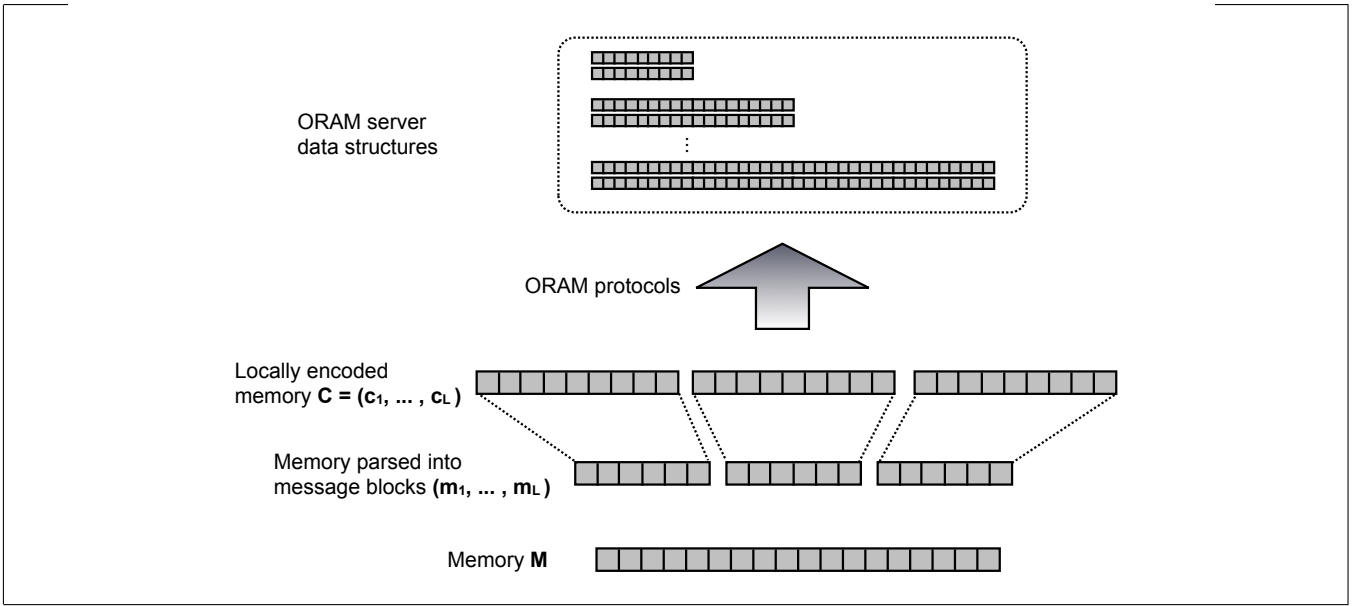


Figure 1: Our Construction

$\mathbf{M} \in \Sigma^\ell$ which she splits into small message blocks $\mathbf{M} = (\mathbf{m}_1, \dots, \mathbf{m}_L)$ with $\mathbf{m}_i \in \Sigma^k$ where the block size $k \ll \ell = Lk$ is only dependant on the security parameter. She then applies an error correcting code $\text{Enc} : \Sigma^k \rightarrow \Sigma^n$ that can efficiently recover $\frac{n}{2}$ erasures to each message block individually, resulting in the value $\mathbf{C} = (\mathbf{c}_1, \dots, \mathbf{c}_L) \in \Sigma^{Ln}$ where $\mathbf{c}_i = \text{Enc}(\mathbf{m}_i)$. Finally, she initializes an ORAM scheme with the initial data $\mathbf{D} = \mathbf{C}$, which the ORAM stores on the server in some clever privacy-preserving form, while keeping only a short local state at the client.

Whenever the client wants to read or write to some location within her data, she uses the ORAM scheme to perform the necessary reads/writes on each of the n relevant codeword symbols of \mathbf{C} (see details in Section 5). To run an audit, the client chooses t (\approx security parameter) random locations in $\{1, \dots, Ln\}$ and runs the ORAM read protocol t times to read the corresponding symbols of \mathbf{C} that reside in these locations, checking them for authenticity.

Catching Disregarded Updates. First, let us start with a sanity check, to explain how the above construction can thwart a specific attack in which the server simply disregards the latest update. In particular, such attack should be caught by a subsequent audit. During the audit, the client runs the ORAM protocol to read t random codeword symbols and these are *unlikely* to coincide with any of the n codeword symbols modified by the latest update (recall that t and n are both small and independent of the data size ℓ). However, the ORAM scheme stores data on the server in a highly organized data-structure, and ensures that the most recently updated data is accessed during *any* subsequent “read” execution, even for an unrelated logical location. This is implied by ORAM security since we need to hide whether or not the location of a read was recently updated or not. Therefore, although the audit executes the “ORAM read” protocols on random logical locations inside \mathbf{C} , the ORAM scheme will end up scanning recently updated areas of the server’s actual storage and check them for authenticity, ensuring that recent updates have not been disregarded.

Security and “Next-Read Pattern Hiding”. The high-level security intuition for our PoR scheme is quite simple. The ORAM hides from the server where the various locations of \mathbf{C} reside in its storage, even after observing the access pattern of read/write executions. Therefore it is difficult for the server to reach a state where it will fail on read executions for most locations within some single codeword block (lose data) without also failing on too many read executions altogether (lose the ability to pass an audit).

Making the above intuition formal is quite subtle, and it turns out that standard notion of ORAM

security does *not* suffice. The main issue is that the server may be able to somehow delete *all* (or most) of the n codeword symbols that fall within *some* codeword block $\mathbf{c}_i = (\mathbf{C}[j+1], \dots, \mathbf{C}[j+n])$ without knowing *which* block it deleted. Therefore, although the server will fail on any subsequent read if and only if its location falls within the range $\{j+1, \dots, j+n\}$, it will not learn anything about the location of the read itself since it does not know the index j . Indeed, we will give an example of a contrived ORAM scheme where such an attack is possible and our resulting construction of PoR using this ORAM is *insecure*.

We show, however, that the intuitive reasoning above can be salvaged if the ORAM scheme achieves a new notion of security that we call *next-read pattern hiding (NRPH)*, which may be of independent interest. NRPH security considers an adversarial server that first gets to observe many read/write protocol executions performed sequentially with the client, resulting in some final client configuration \mathcal{C}_{fin} . The adversarial server then gets to see various possibilities for how the “next read” operation would be executed by the client for various distinct locations, where each such execution starts from the same *fixed* client configuration \mathcal{C}_{fin} .⁵ The server should not be able to discern any *relationship* between these executions and the locations they are reading. For example, two such “next-read” executions where the client reads two consecutive locations should be indistinguishable from two executions that read two random and unrelated locations. This notion of NRPH security will be used to show that server cannot reach a state where it can *selectively* fail to respond on read queries whose location falls within some small range of a single codeword block (lose data), but still respond correctly to most completely random reads (pass an audit).

Proving Security via an Extractor. As mentioned earlier, the security of PoR is formalized via an extractor and we now give a high-level overview of how such an extractor works. In particular, we claim that we can take any adversarial server that has a “good” chance of passing an audit and use the extractor to efficiently recover the latest version of the client data from it. The extractor initializes an “empty array” \mathbf{C} . It then executes random audit protocols with the server, by acting as the honest client. In particular, it chooses t random locations within the array and runs the corresponding ORAM read protocols. If the execution of the audit is successful, the extractor fills in the corresponding values of \mathbf{C} that it learned during the audit execution. In either case, it then rewinds the server and runs a fresh execution of the audit, repeating this step for several iterations.

Since the server has a good chance of passing a random audit, it is easy to show that the extractor can eventually recover a large fraction, say $> \frac{3}{4}$, of the entries inside \mathbf{C} by repeating this process sufficiently many times. Because of the *authenticity* of the ORAM, the recovered values are the correct ones, corresponding to the latest version of the client data. Now we need to argue that there is no codeword block \mathbf{c}_i within \mathbf{C} for which the extractor recovered fewer than $\frac{1}{2}$ of its codeword symbols, as this would prevent us from applying erasure decoding and recovering the underlying message block. Let FAILURE denote the above bad event. If all the recovered locations (comprising $> \frac{3}{4}$ fraction of the total) were distributed uniformly within \mathbf{C} then FAILURE would occur with negligible probability, as long as the codeword size n is sufficiently large in the security parameter. We can now rely on the NRPH security of the ORAM to ensure that FAILURE also happens with negligible probability in our case. We can think of the FAILURE event as a function of the locations queried by the extractor in each audit execution, and the set of executions on which the server fails. If the malicious server can cause FAILURE to occur, it means that it can distinguish the pattern of locations actually queried by the extractor during the audit executions (for which the FAILURE event occurs) from a randomly permuted pattern of locations (for which the FAILURE event does not occur with overwhelming probability). Note that the use of rewinding between the audit executions of the extractor forces us to rely on NRPH security rather than just standard ORAM security.

The above presents the high-level intuition and is somewhat oversimplified. See Section 4 for the formal definition of NRPH security and Section 5 for the formal description of our dynamic PoR scheme and a rigorous proof of security.

⁵This is in contrast to the standard sequential operations where the client state is updated after each execution.

Achieving Next-Read Pattern Hiding. We show that standard ORAM security does *not* generically imply NRPH security, by giving a contrived scheme that satisfies the former but not the latter. Nevertheless, many natural ORAM constructions in the literature *do* seem to satisfy NRPH security. In particular, we examine the efficient ORAM construction of Goodrich and Mitzenmacher [15] and prove that (with minor modifications) it is NRPH secure.

Contributions. We call our final scheme PORAM since it combines the techniques and security of PoR and ORAM. In particular, other than providing provable dynamic cloud storage as was our main goal, our scheme also satisfies the strong *privacy* guarantees of ORAM, meaning that it hides all contents of the remotely stored data as well as the access pattern of which locations are accessed when. It also provides strong *authenticity* guarantees (same as *memory checking*; see Section 1.2), ensuring that any “read” execution with a malicious remote server is guaranteed to return the latest version of the data (or detect cheating).

In brief, our contributions can be summarized as follows:

- We give the first asymptotically efficient solution to PoR for outsourced dynamic data, where a successful audit ensures that the server knows the latest version of the client data. In particular:
 - Client storage is small and independent of the data size.
 - Server storage is linear in the data size, expanding it by only a small constant factor.
 - Communication and computation of client and server during *read*, *write*, and *audit* executions are polylogarithmic in the size of the client data.
- Our scheme also achieves strong *privacy* and *authenticity* guarantees, matching those of *oblivious RAM* and *memory checking*.
- We present a new security notion called “next-read pattern hiding (NRPH)” for ORAM and a construction achieving this new notion, which may be of independent interest.

We mention that the PORAM scheme is simple to implement and has low concrete efficiency overhead *on top of* an underlying ORAM scheme with NRPH security. There is much recent and ongoing research activity in instantiating/implementing truly practical ORAM schemes, which are likely to yield correspondingly practical instantiations of our PORAM protocol.

1.2 Related Work

Proofs of retrievability for *static* data were initially defined and constructed by Juels and Kaliski [19], building on a closely related notion called sublinear-authenticators of Naor and Rothblum [23]. Concurrently, Ateniese et al. [1] defined another related primitive called *provable data possession* (PDP). Since then, there has been much ongoing research activity on PoR and PDP schemes.

PoR vs. PDP. The main difference between PoR and PDP is the notion of security that they achieve. A PoR audit guarantees that the server maintains knowledge of *all* of the client data, while a PDP audit only ensures that the server is storing *most* of the client data. For example, in a PDP scheme, the server may lose a small portion of client data (say 1 MB out of a 10 GB file) and may maintain an high chance of passing a future audit.⁶ On a technical level, the main difference in most prior PDP/PoR constructions is that PoR schemes store a *redundant encoding* of the client data on the server. For a detailed comparison, see Küpgü [21, 22].

⁶An alternative way to use PDPs can also achieve full security, at the cost of requiring the server to read the entire client data during an audit, but still minimizing the communication complexity. If the data is large, say 10 GB, this is vastly impractical.

Static Data. PoR and PDP schemes for static data (without updates) have received much research attention [27, 10, 7, 2], with works improving on communication efficiency and exact security, yielding essentially optimal solutions. Another interesting direction has been to extend these works to the multi-server setting [6, 8, 9] where the client can use the audit mechanism to identify faulty machines and recover the data from the others.

Dynamic Data. The works of Ateniese et al. [3], Erway et al. [12] and Wang et al. [30] show how to achieve PDP security for *dynamic data*, supporting efficient updates. This is closely related to work on memory checking [5, 23, 11], which studies how to authenticate remotely stored dynamic data so as to allow efficient reads/writes, while being able to verify the authenticity of the latest version of the data (preventing the server from “rolling back” updates and using an old version). Unfortunately, these techniques alone cannot be used to achieve the stronger notion of PoR security. Indeed, the main difficulty that we resolve in this work, how to efficiently update *redundantly encoded data*, does not come up in the context of PDP.

A recent work of Stefanov et al. [29] considers PoR for dynamic data, but in a more complex setting where an additional trusted “portal” performs some operations on behalf of the client, and can cache updates for an extended period of time. It is not clear if these techniques can be translated to the basic client/server setting, which we consider here. However, even in this modified setting, the complexity of the updates and the audit in that work is proportional to *square-root* of the data size, whereas ours is *polylogarithmic*.

2 Preliminaries

Notation. Throughout, we use λ to denote the *security parameter*. We identify *efficient* algorithms as those running in (probabilistic) polynomial time in λ and their input lengths, and identify *negligible* quantities (e.g., acceptable error probabilities) as $\text{negl}(\lambda) = 1/\lambda^{\omega(1)}$, meaning that they are asymptotically smaller than $1/\lambda^c$ for every constant $c > 0$. For $n \in \mathbb{N}$, we define the set $[n] \stackrel{\text{def}}{=} \{1, \dots, n\}$. We use the notation $(k \bmod n)$ to denote the unique integer $i \in \{0, \dots, n-1\}$ such that $i \equiv k \pmod{n}$.

Erasur Codes. We say that (Enc, Dec) is an $(n, k, d)_\Sigma$ -code with *efficient erasure decoding* over an alphabet Σ if the original message can always be recovered from a corrupted codeword with at most $d-1$ erasures. That is, for every *message* $\mathbf{m} = (m_1, \dots, m_k) \in \Sigma^k$ giving a *codeword* $\mathbf{c} = (c_1, \dots, c_n) = \text{Enc}(\mathbf{m})$, and every corrupted codeword $\tilde{\mathbf{c}} = (\tilde{c}_1, \dots, \tilde{c}_n)$ such that $\tilde{c}_i \in \{c_i, \perp\}$ and the number of erasures is $|\{i \in [n] : \tilde{c}_i = \perp\}| \leq d-1$, we have $\text{Dec}(\tilde{\mathbf{c}}) = \mathbf{m}$. We say that a code is *systematic* if, for every message \mathbf{m} , the codeword $\mathbf{c} = \text{Enc}(\mathbf{m})$ contains \mathbf{m} in the first k positions $c_1 = m_1, \dots, c_k = m_k$. A systematic variant of the Reed-Solomon code achieves the above for any integers $n > k$ and any *field* Σ of size $|\Sigma| \geq n$ with $d = n - k + 1$.

Virtual Memory. We think of *virtual memory* \mathbf{M} , with *word-size* w and *length* ℓ , as an array $\mathbf{M} \in \Sigma^\ell$ where $\Sigma \stackrel{\text{def}}{=} \{0, 1\}^w$. We assume that, initially, each location $\mathbf{M}[i]$ contains the special *uninitialized symbol* $\mathbf{0} = 0^w$. Throughout, we will think of ℓ as some large polynomial in the security parameter, which upper bounds the amount of memory that can be used.

Outsourcing Virtual Memory. In the next two sections, we look at two primitives: *dynamic PoR* and *ORAM*. These primitives allow a client to *outsource* some virtual memory \mathbf{M} to a remote server, while providing useful security guarantees. Reading and writing to some location of \mathbf{M} now takes on the form of a protocol execution with the server. The goal is to provide security while preserving efficiency in terms of client/server computation, communication, and the number of server-memory accesses per operation, which should all be *poly-logarithmic* in the length ℓ . We also want to optimize the size of the client storage (independent of ℓ) and server storage (not much larger than ℓ). We find this abstract view of outsourcing memory to be the simplest and most general to work with. Any higher-level data-structures and operations

(e.g., allowing appends/inserts to data or implementing an entire file-system) can be easily done *on top of* this abstract notion of memory and therefore securely outsourced to the remote server.

3 Dynamic PoR

A *Dynamic PoR* scheme consists of protocols **Plnit**, **PRead**, **PWrite**, **Audit** between two *stateful* parties: a client \mathcal{C} and a server \mathcal{S} . The server acts as the curator for some virtual memory \mathbf{M} , which the client can *read*, *write* and *audit* by initiating the corresponding interactive protocols:

- **Plnit**($1^\lambda, 1^w, \ell$): This protocol corresponds to the client initializing an (empty) virtual memory \mathbf{M} with word-size w and length ℓ , which it supplies as inputs.
- **PRead**(i): This protocol corresponds to the client reading $v = \mathbf{M}[i]$, where it supplies the input i and outputs some value v at the end.
- **PWrite**(i, v): This protocol corresponds to setting $\mathbf{M}[i] := v$, where the client supplies the inputs i, v .
- **Audit**: This protocol is used by the client to verify that the server is maintaining the memory contents correctly so that they remain retrievable. The client outputs a decision $b \in \{\text{accept}, \text{reject}\}$.

The client \mathcal{C} in the protocols may be *randomized*, but we assume (w.l.o.g.) that the honest server \mathcal{S} is deterministic. At the conclusion of the **Plnit** protocol, both the client and the server create some long-term local state, which each party will update during the execution of each of the subsequent protocols. The client may also output **reject** during the execution of the **Plnit**, **PRead**, **PWrite** protocols, to denote that it detected some misbehavior of the server. Note that we assume that the virtual memory is initially *empty*, but if the client has some initial data, she can write it onto the server block-by-block immediately after initialization. For ease of presentation, we may assume that the state of the client and the server always contains the security parameter, and the memory parameters $(1^\lambda, 1^w, \ell)$.

We now define the three properties of a dynamic PoR scheme: *correctness*, *authenticity* and *retrievability*. For these definitions, we say that $P = (op_0, op_1, \dots, op_q)$ is a dynamic PoR *protocol sequence* if $op_0 = \mathbf{Plnit}(1^\lambda, 1^w, \ell)$ and, for $j > 0$, $op_j \in \{\mathbf{PRead}(i), \mathbf{PWrite}(i, v), \mathbf{Audit}\}$ for some index $i \in [\ell]$ and value $v \in \{0, 1\}^w$.

Correctness. If the client and the server are both *honest* and $P = (op_0, \dots, op_q)$ is some protocol sequence, then we require the following to occur with probability 1 over the randomness of the client:

- Each execution of a protocol $op_j = \mathbf{PRead}(i)$ results in the client outputting the correct value $v = \mathbf{M}[i]$, matching what would happen if the corresponding operations were performed directly on a memory \mathbf{M} . In particular, v is the value contained in the most recent prior write operation with location i , or, if no such prior operation exists, $v = \mathbf{0}$.
- Each execution of the **Audit** protocol results in the decision $b = \text{accept}$.

Authenticity. We require that the client can always *detect* if any protocol message sent by the server deviates from honest behavior. More precisely, consider the following game $\text{AuthGame}_{\tilde{\mathcal{S}}}(\lambda)$ between a malicious server $\tilde{\mathcal{S}}$ and a challenger:

- The malicious server $\tilde{\mathcal{S}}(1^\lambda)$ specifies a valid protocol sequence $P = (op_0, \dots, op_q)$.
- The challenger initializes a copy of the honest client \mathcal{C} and the (deterministic) honest server \mathcal{S} . It sequentially executes op_0, \dots, op_q between \mathcal{C} and the malicious server $\tilde{\mathcal{S}}$ while, in parallel, also passing a copy of every message from \mathcal{C} to the honest server \mathcal{S} .
- If, at any point during the execution of some op_j , any protocol message given by $\tilde{\mathcal{S}}$ differs from that of \mathcal{S} , and the client \mathcal{C} does not output **reject**, the adversary wins and the game outputs 1. Else 0.

For any efficient adversarial server $\tilde{\mathcal{S}}$, we require $\Pr[\text{AuthGame}_{\tilde{\mathcal{S}}}(\lambda) = 1] \leq \text{negl}(\lambda)$. Note that authenticity and correctness together imply that the client will always either read the correct value corresponding to the latest contents of the virtual memory or reject whenever interacting with a malicious server.

Retrievability. Finally we define the main purpose of a dynamic PoR scheme, which is to ensure that the client data remains retrievable. We wish to guarantee that, whenever the malicious server is in a state with a reasonable probability δ of successfully passing an audit, he must *know* the entire content of the client’s virtual memory \mathbf{M} . As in “proofs of knowledge”, we formalize *knowledge* via the existence of an efficient *extractor* \mathcal{E} which can recover the value \mathbf{M} given (black-box) access to the malicious server.

More precisely, we define the game $\text{ExtGame}_{\tilde{\mathcal{S}}, \mathcal{E}}(\lambda, p)$ between a malicious server $\tilde{\mathcal{S}}$, extractor \mathcal{E} , and challenger:

- The malicious server $\tilde{\mathcal{S}}(1^\lambda)$ specifies a protocol sequence $P = (op_0, \dots, op_q)$. Let $\mathbf{M} \in \Sigma^\ell$ be the correct value of the memory contents at the end of executing P .
- The challenger initializes a copy of the honest client \mathcal{C} and sequentially executes op_0, \dots, op_q between \mathcal{C} and $\tilde{\mathcal{S}}$. Let \mathcal{C}_{fin} and $\tilde{\mathcal{S}}_{\text{fin}}$ be the final configurations (states) of the client and malicious server at the end of this interaction, including all of the random coins of the malicious server. Define the success-probability

$$\text{Succ}(\tilde{\mathcal{S}}_{\text{fin}}) \stackrel{\text{def}}{=} \Pr \left[\tilde{\mathcal{S}}_{\text{fin}} \xrightarrow{\text{Audit}} \mathcal{C}_{\text{fin}} = \text{accept} \right]$$

as the probability that an execution of a subsequent **Audit** protocol between $\tilde{\mathcal{S}}_{\text{fin}}$ and \mathcal{C}_{fin} results in the latter outputting **accept**. The probability is only over the random coins of \mathcal{C}_{fin} during this execution.

- Run $\mathbf{M}' \leftarrow \mathcal{E}^{\tilde{\mathcal{S}}_{\text{fin}}}(\mathcal{C}_{\text{fin}}, 1^\ell, 1^p)$, where the extractor \mathcal{E} gets *black-box rewinding access* to the malicious server in its final configuration $\tilde{\mathcal{S}}_{\text{fin}}$, and attempts to extract out the memory contents as \mathbf{M}' .⁷
- If $\text{Succ}(\tilde{\mathcal{S}}_{\text{fin}}) \geq 1/p$ and $\mathbf{M}' \neq \mathbf{M}$ then output 1, else 0.

We require that there exists a probabilistic-poly-time extractor \mathcal{E} such that, for every efficient malicious server $\tilde{\mathcal{S}}$ and every polynomial $p = p(\lambda)$ we have $\Pr[\text{ExtGame}_{\tilde{\mathcal{S}}, \mathcal{E}}(\lambda, p) = 1] \leq \text{negl}(\lambda)$.

The above says that whenever the malicious server reaches some state $\tilde{\mathcal{S}}_{\text{fin}}$ in which it maintains a $\delta \geq 1/p$ probability of passing the *next audit*, the extractor \mathcal{E} will be able to extract out the correct memory contents \mathbf{M} from $\tilde{\mathcal{S}}_{\text{fin}}$, meaning that the server must retain full *knowledge* of \mathbf{M} in this state. The extractor is efficient, but can run in time polynomial in p and the size of the memory ℓ .

A Note on Adaptivity. We defined the above *authenticity* and *retrievability* properties assuming that the sequence of read/write operations is adversarial, but is chosen *non-adaptively*, before the adversarial server sees any protocol executions. This seems to be sufficient in most realistic scenarios, where the server is unlikely to have any influence on which operations the client wants to perform. It also matches the security notions in prior works on ORAM. Nevertheless, we note that our final results also achieve adaptive security, where the attacker can choose the sequence of operations op_i adaptively after seeing the execution of previous operations, if the underlying ORAM satisfies this notion. Indeed, most prior ORAM solutions seem to do so, but it was never included in their analysis.

4 Oblivious RAM with Next-Read Pattern Hiding

An ORAM consists of protocols (**OInit**, **ORead**, **OWrite**) between a client \mathcal{C} and a server \mathcal{S} , with the same syntax as the corresponding protocols in PoR. We will also extend the syntax of **ORead** and **OWrite** to allow for reading/writing from/to multiple distinct locations simultaneously. That is, for arbitrary $t \in \mathbb{N}$, we define

⁷This is similar to the extractor in zero-knowledge proofs of knowledge. In particular \mathcal{E} can execute protocols with the malicious server in its state $\tilde{\mathcal{S}}_{\text{fin}}$ and rewind it back this state at the end of the execution.

the protocol $\mathbf{ORed}(i_1, \dots, i_t)$ for *distinct* indices $i_1, \dots, i_t \in [\ell]$, in which the client outputs (v_1, \dots, v_t) corresponding to reading $v_1 = \mathbf{M}[i_1], \dots, v_t = \mathbf{M}[i_t]$. Similarly, we define the protocol $\mathbf{OWrite}(i_t, \dots, i_1; v_1, \dots, v_t)$ for *distinct* indices $i_1, \dots, i_t \in [\ell]$, which corresponds to setting $\mathbf{M}[i_1] := v_1, \dots, \mathbf{M}[i_t] := v_t$.

We say that $P = (op_0, \dots, op_q)$ is an *ORAM protocol sequence* if $op_0 = \mathbf{OInit}(1^\lambda, 1^w, \ell)$ and, for $j > 0$, op_j is a valid (multi-location) read/write operation.

We require that an ORAM construction needs to satisfy *correctness* and *authenticity*, which are defined the same way as in PoR.⁸ For privacy, we define a new property called *next-read pattern hiding*. For completeness, we also define the standard notion of ORAM pattern hiding in Appendix B.

Next-Read Pattern Hiding. Consider an *honest-but-curious* server \mathcal{A} who observes the execution of some protocol sequence P with a client \mathcal{C} resulting in the final client configuration \mathcal{C}_{fin} . At the end of this execution, \mathcal{A} gets to observe how \mathcal{C}_{fin} would execute the *next* read operation $\mathbf{ORed}(i_1, \dots, i_t)$ for various different t -tuples (i_1, \dots, i_t) of locations, but always starting in the same client state \mathcal{C}_{fin} . We require that \mathcal{A} cannot observe any correlation between these next-read executions and their locations, up to *equality*. That is, \mathcal{A} should not be able to distinguish if \mathcal{C}_{fin} instead executes the next-read operations on *permuted locations* $\mathbf{ORed}(\pi(i_1), \dots, \pi(i_t))$ for a permutation $\pi : [\ell] \rightarrow [\ell]$.

More formally, we define $\text{NextReadGame}_{\mathcal{A}}^b(\lambda)$, for $b \in \{0, 1\}$, between an adversary \mathcal{A} and a challenger:

- The attacker $\mathcal{A}(1^\lambda)$ chooses an ORAM protocol sequence $P_1 = (op_0, \dots, op_{q_1})$. It also chooses a sequence $P_2 = (rop_1, \dots, rop_{q_2})$ of valid multi-location read operations, where each operation is of the form $rop_j = \mathbf{ORed}(i_{j,1}, \dots, i_{j,t_j})$ with t_j distinct locations. Lastly, it chooses a permutation $\pi : [\ell] \rightarrow [\ell]$. For each rop_j in P_2 , define a permuted version $rop'_j := \mathbf{ORed}(\pi(i_{j,1}), \dots, \pi(i_{j,t_j}))$. The game now proceeds in two stages.
- *Stage I.* The challenger initializes the honest client \mathcal{C} and the (deterministic) honest server \mathcal{S} . It sequentially executes the protocols $P = (op_0, \dots, op_{q_1})$ between \mathcal{C} and \mathcal{S} . Let $\mathcal{C}_{\text{fin}}, \mathcal{S}_{\text{fin}}$ be the final configuration of the client and server at the end.
- *Stage II.* For each $j \in [q_2]$: challenger either executes the original operation rop_j if $b = 0$, or the permuted operation rop'_j if $b = 1$, between \mathcal{C} and \mathcal{S} . At the end of each operation execution it resets the configuration of the client and server back to $\mathcal{C}_{\text{fin}}, \mathcal{S}_{\text{fin}}$ respectively, before the next execution.
- The adversary \mathcal{A} is given the transcript of all the protocol executions in stages I and II, and outputs a bit \tilde{b} which we define as the output of the game. Note that, since the honest server \mathcal{S} is deterministic, seeing the protocol transcripts between \mathcal{S} and \mathcal{C} is the same as seeing the entire internal state of \mathcal{S} at any point time.

We require that, for every efficient \mathcal{A} , we have

$$|\Pr[\text{NextReadGame}_{\mathcal{A}}^0(\lambda) = 1] - \Pr[\text{NextReadGame}_{\mathcal{A}}^1(\lambda) = 1]| \leq \text{negl}(\lambda).$$

5 PORAM: Dynamic PoR via ORAM

We now give our construction of dynamic PoR, using ORAM. Since the ORAM security properties are preserved by the construction as well, we happen to achieve ORAM and dynamic PoR simultaneously. Therefore, we call our construction PORAM.

⁸Traditionally, authenticity is not always defined/required for ORAM. However, it is crucial for our use. As noted in several prior works, it can often be added at almost no cost to efficiency. It can also be added generically by running a *memory checking* scheme on top of ORAM. See Section 6.4 for details.

Overview of Construction. Let (Enc, Dec) be an $(n, k, d = n - k + 1)_\Sigma$ systematic code with efficient erasure decoding over the alphabet $\Sigma = \{0, 1\}^w$ (e.g., the systematic Reed-Solomon code over \mathbb{F}_{2^w}). Our construction of dynamic PoR will interpret the memory $\mathbf{M} \in \Sigma^\ell$ as consisting of $L = \ell/k$ consecutive *message blocks*, each having k alphabet symbols (assume k is small and divides ℓ). The construction implicitly maps operation on \mathbf{M} to operations on *encoded memory* $\mathbf{C} \in (\Sigma)^{\ell_{\text{code}}=Ln}$, which consists of L *codeword blocks* with n alphabet symbols each. The L codeword blocks $\mathbf{C} = (\mathbf{c}_1, \dots, \mathbf{c}_L)$ are simply the encoded versions of the corresponding message blocks in $\mathbf{M} = (\mathbf{m}_1, \dots, \mathbf{m}_L)$ with $\mathbf{c}_q = \text{Enc}(\mathbf{m}_q)$ for $q \in [L]$. This means that, for each $i \in [\ell]$, the value of the memory location $\mathbf{M}[i]$ can only affect the values of the encoded-memory locations $\mathbf{C}[j+1], \dots, \mathbf{C}[j+n]$ where $j = n \cdot \lfloor i/k \rfloor$. Furthermore, since the encoding is *systematic*, we have $\mathbf{M}[i] = \mathbf{C}[j+u]$ where $u = (i \bmod k) + 1$. To read the memory location $\mathbf{M}[i]$, the client will use ORAM to read the codeword location $\mathbf{C}[j+u]$. To write to the memory location $\mathbf{M}[i] := v$, the client needs to update the entire corresponding codeword block. She does so by first using ORAM to read the corresponding codeword block $\mathbf{c} = (\mathbf{C}[j+1], \dots, \mathbf{C}[j+n])$, and decodes to obtain the original memory block $\mathbf{m} = \text{Dec}(\mathbf{c})$.⁹ She then locally updates the memory block by setting $\mathbf{m}[u] := v$, re-encodes the updated memory block to get $\mathbf{c}' = (c'_1, \dots, c'_n) := \text{Enc}(\mathbf{m})$ and uses the ORAM to write \mathbf{c}' back into the encoded memory, setting $\mathbf{C}[j+1] := c'_1, \dots, \mathbf{C}[j+n] := c'_n$.

The Construction. Our PORAM construction is defined for some parameters $n > k, t \in \mathbb{N}$. Let $\mathbf{O} = (\mathbf{O}\text{Init}, \mathbf{O}\text{Read}, \mathbf{O}\text{Write})$ be an ORAM. Let (Enc, Dec) be an $(n, k, d = n - k + 1)_\Sigma$ systematic code with efficient erasure decoding over the alphabet $\Sigma = \{0, 1\}^w$ (e.g., the systematic Reed-Solomon code over \mathbb{F}_{2^w}).

- **PInit** $(1^\lambda, 1^w, \ell)$: Assume k divides ℓ and let $\ell_{\text{code}} := n \cdot (\ell/k)$. Run the $\mathbf{O}\text{Init}(1^\lambda, 1^w, \ell_{\text{code}})$ protocol.
- **PRead** (i) : Let $i' := n \cdot \lfloor i/k \rfloor + (i \bmod k) + 1$ and run the $\mathbf{O}\text{Read}(i')$ protocol.
- **PWrite** (i, v) : Set $j := n \cdot \lfloor i/k \rfloor$ and $u := (i \bmod k) + 1$.
 - Run $\mathbf{O}\text{Read}(j+1, \dots, j+n)$ and get output $\mathbf{c} = (c_1, \dots, c_n)$.
 - Decode $\mathbf{m} = (m_1, \dots, m_k) = \text{Dec}(\mathbf{c})$.
 - Modify position u of \mathbf{m} by locally setting $m_u := v$. Re-encode the modified message-block \mathbf{m} by setting $\mathbf{c}' = (c'_1, \dots, c'_n) := \text{Enc}(\mathbf{m})$.
 - Run $\mathbf{O}\text{Write}(j+1, \dots, j+n; c'_1, \dots, c'_n)$.
- **Audit**: Pick t distinct indices $j_1, \dots, j_t \in [\ell_{\text{code}}]$ at random. Run $\mathbf{O}\text{Read}(j_1, \dots, j_t)$ and return **accept** iff the protocol finished without outputting **reject**.

If, any ORAM protocol execution in the above scheme outputs **reject**, the client enters a special rejection state in which it stops responding and automatically outputs **reject** for any subsequent protocol execution.

It is easy to see that if the underlying ORAM scheme used in the above PORAM construction is secure in the standard sense of ORAM (see Appendix B) then the above construction preserves this ORAM security, hiding which locations are being accessed in each operation. As our main result, we now prove that if the ORAM scheme satisfies next-read pattern hiding (NRPH) security then the PORAM construction above is also a secure dynamic PoR scheme.

Theorem 1. *Assume that $\mathbf{O} = (\mathbf{O}\text{Init}, \mathbf{O}\text{Read}, \mathbf{O}\text{Write})$ is an ORAM with next-read pattern hiding (NRPH) security, and we choose parameters $k = \Omega(\lambda)$, $k/n = (1 - \Omega(1))$, $t = \Omega(\lambda)$. Then the above scheme $\text{PORAM} = (\text{PInit}, \text{PRead}, \text{PWrite}, \text{Audit})$ is a dynamic PoR scheme.*

⁹We can skip this step if the client already has the value \mathbf{m} stored locally e.g. from prior read executions.

5.1 Proof of Theorem 1

The correctness and authenticity properties of PORAM follow immediately from those of the underlying ORAM scheme **O**. The main challenge is to show that the *retrievability* property holds. As a first step, let us describe the extractor.

The Extractor. The extractor $\mathcal{E}^{\tilde{\mathcal{S}}_{\text{fin}}}(\mathcal{C}_{\text{fin}}, 1^\ell, 1^p)$ works as follows:

- (1) Initialize $\mathbf{C} := (\perp)^{\ell_{\text{code}}}$ where $\ell_{\text{code}} = n(\ell/k)$ to be an empty vector.
- (2) Keep rewinding and auditing the server by repeating the following step for $s = \max(2\ell_{\text{code}}, \lambda) \cdot p$ times: Pick t distinct indices $j_1, \dots, j_t \in [\ell_{\text{code}}]$ at random and run the protocol $\mathbf{ORed}(j_1, \dots, j_t)$ with $\tilde{\mathcal{S}}_{\text{fin}}$, acting as \mathcal{C}_{fin} as in the audit protocol. If the protocol is accepting and \mathcal{C}_{fin} outputs (v_1, \dots, v_t) , set $\mathbf{C}[j_1] := v_1, \dots, \mathbf{C}[j_t] := v_t$. Rewind $\tilde{\mathcal{S}}_{\text{fin}}, \mathcal{C}_{\text{fin}}$ to their state prior to this execution for the next iteration.
- (3) Let $\delta \stackrel{\text{def}}{=} (1 + \frac{k}{n})/2$. If the number of “filled in” values in \mathbf{C} is $|\{j \in [\ell_{\text{code}}] : \mathbf{C}[j] \neq \perp\}| < \delta \cdot \ell_{\text{code}}$ then output fail_1 . Else interpret \mathbf{C} as consisting of $L = \ell/k$ consecutive codeword blocks $\mathbf{C} = (\mathbf{c}_1, \dots, \mathbf{c}_L)$ with each block $\mathbf{c}_j \in \Sigma^n$. If there exists some index $j \in [L]$ such that the number of “filled” in values in codeword block \mathbf{c}_j is $|\{i \in [n] : \mathbf{c}_j[i] \neq \perp\}| < k$ then output fail_2 . Otherwise, apply erasure decoding to each codeword block \mathbf{c}_j , to recover $\mathbf{m}_j = \text{Dec}(\mathbf{c}_j)$, and output $\mathbf{M} = (\mathbf{m}_1, \dots, \mathbf{m}_L) \in \Sigma^\ell$.¹⁰

Proof by Contradiction. Assume that PORAM does *not* satisfy the retrievability property with the above extractor \mathcal{E} . Then there exists some efficient adversarial server $\tilde{\mathcal{S}}$ and some polynomials $p = p(\lambda), p' = p'(\lambda)$ such that, for infinitely many values $\lambda \in \mathbb{N}$, we have:

$$\Pr[\text{ExtGame}_{\tilde{\mathcal{S}}, \mathcal{E}}(\lambda, p(\lambda)) = 1] > \frac{1}{p'(\lambda)} \quad (1)$$

Using the same notation as in the definition of ExtGame , let $\tilde{\mathcal{S}}_{\text{fin}}, \mathcal{C}_{\text{fin}}$ be the final configurations of the malicious server $\tilde{\mathcal{S}}$ and client \mathcal{C} , respectively, after executing the protocol sequence P chosen by the server at the beginning of the game, and let \mathbf{M} be the correct value of the memory contents resulting from P . Then (1) implies

$$\Pr \left[\begin{array}{l} \text{Succ}(\tilde{\mathcal{S}}_{\text{fin}}) > \frac{1}{p(\lambda)} \\ \wedge \mathcal{E}^{\tilde{\mathcal{S}}_{\text{fin}}}(\mathcal{C}_{\text{fin}}, 1^\ell, 1^p) \neq \mathbf{M} \end{array} \right] > \frac{1}{p'(\lambda)} \quad (2)$$

where the probability is over the coins of $\mathcal{C}, \tilde{\mathcal{S}}$ which determine the final configuration $\tilde{\mathcal{S}}_{\text{fin}}, \mathcal{C}_{\text{fin}}$ and the coins of the extractor \mathcal{E} . We now slowly refine the above inequality until we reach a contradiction, showing that the above cannot hold.

Extractor can only fail with $\{\text{fail}_1, \text{fail}_2\}$. Firstly, we argue that at the conclusion of ExtGame , the extractor must either output the correct memory contents \mathbf{M} or must fail with one of the error messages $\{\text{fail}_1, \text{fail}_2\}$. In other words, it can always detect *failure* and never outputs an incorrect value $\mathbf{M}' \neq \mathbf{M}$. This follows from the *authenticity* of the underlying ORAM scheme which guarantees that the extractor never puts any incorrect value into the array \mathbf{C} .

Lemma 1. *Within the execution of $\text{ExtGame}_{\tilde{\mathcal{S}}, \mathcal{E}}(\lambda, p)$, we have:*

$$\Pr[\mathcal{E}^{\tilde{\mathcal{S}}_{\text{fin}}}(\mathcal{C}_{\text{fin}}, 1^\ell, 1^p) \notin \{\mathbf{M}, \text{fail}_1, \text{fail}_2\}] \leq \text{negl}(\lambda).$$

Proof of Lemma. The only way that the above bad event can occur is if the extractor puts an incorrect value into its array \mathbf{C} which does not match encoded version of the correct memory contents \mathbf{M} . In particular, this

¹⁰The failure event fail_1 and the choice of δ is only intended to simplify the analysis of the extractor. The only real bad event from which the extractor cannot recover is fail_2 .

means that one of the audit protocol executions (consisting of an **ORed** with t random locations) initiated by the extractor \mathcal{E} between the malicious server $\tilde{\mathcal{S}}_{\text{fin}}$ and the client \mathcal{C}_{fin} causes the client to output some incorrect value which does not match correct memory contents \mathbf{M} , and *not* reject. By the *correctness* of the ORAM scheme, this means that the malicious server must have deviated from honest behavior during that protocol execution, without the client rejecting. Assume the probability of this bad event happening is ρ . Since the extractor runs $s = \max(2\ell_{\text{code}}, \lambda) \cdot p = \text{poly}(\lambda)$ such protocol executions *with rewinding*, there is at least $\rho/s = \rho/\text{poly}(\lambda)$ probability that the above bad event occurs on a single random execution of the audit with $\tilde{\mathcal{S}}_{\text{fin}}$. But this means that $\tilde{\mathcal{S}}$ can be used to break the *authenticity of ORAM* with advantage $\rho/\text{poly}(\lambda)$, by first running the requested protocol sequence P and then deviating from honest behavior during a subsequent **ORed** protocol without being detected. Therefore, by the authenticity of ORAM, we must have $\rho = \text{negl}(\lambda)$. \square

Combining the above with (2) we get:

$$\Pr \left[\begin{array}{l} \text{Succ}(\tilde{\mathcal{S}}_{\text{fin}}) > \frac{1}{p(\lambda)} \\ \wedge \mathcal{E}^{\tilde{\mathcal{S}}_{\text{fin}}}(\mathcal{C}_{\text{fin}}, 1^\ell, 1^p) \in \{\text{fail}_1, \text{fail}_2\} \end{array} \right] > \frac{1}{p'(\lambda)} - \text{negl}(\lambda) \quad (3)$$

Extractor can indeed only fail with fail₂. Next, we refine equation (3) and claim that the extractor is unlikely to reach the failure event fail_1 and therefore must fail with fail_2 .

$$\Pr \left[\begin{array}{l} \text{Succ}(\tilde{\mathcal{S}}_{\text{fin}}) > \frac{1}{p(\lambda)} \\ \wedge \mathcal{E}^{\tilde{\mathcal{S}}_{\text{fin}}}(\mathcal{C}_{\text{fin}}, 1^\ell, 1^p) = \text{fail}_2 \end{array} \right] > \frac{1}{p'(\lambda)} - \text{negl}(\lambda) \quad (4)$$

To prove the above, it suffices to prove the following lemma, which intuitively says that if $\tilde{\mathcal{S}}_{\text{fin}}$ has a good chance of passing an audit, then the extractor must be able to extract sufficiently many values inside \mathbf{C} and hence cannot output fail_1 . Remember that fail_1 occurs if the extractor does not have enough values to recover the whole memory, and fail_2 occurs if the extractor does not have enough values to recover some message block.

Lemma 2. *For any (even inefficient) machine $\tilde{\mathcal{S}}_{\text{fin}}$ and any polynomial $p = p(\lambda)$ we have:*

$$\Pr[\mathcal{E}^{\tilde{\mathcal{S}}_{\text{fin}}}(\mathcal{C}_{\text{fin}}, 1^\ell, 1^p) = \text{fail}_1 \mid \text{Succ}(\tilde{\mathcal{S}}_{\text{fin}}) \geq 1/p] \leq \text{negl}(\lambda).$$

Proof of Lemma. Let E be the bad event that fail_1 occurs. For each iteration $i \in [s]$ within step (2) of the execution of \mathcal{E} let us define:

- X_i to be an indicator random variable that takes on the value $X_i = 1$ iff the **ORed** protocol execution in iteration i does not reject.
- G_i to be a random variable that denotes the subset $\{j \in [\ell_{\text{code}}] : \mathbf{C}[j] \neq \perp\}$ of filled-in positions in the current version of \mathbf{C} at the beginning of iteration i .
- Y_i to be an indicator random variable that takes on the value $Y_i = 1$ iff $|G_i| < \delta \cdot \ell_{\text{code}}$ and all of the locations that \mathcal{E} chooses to read in iteration i happen to satisfy $j_1, \dots, j_t \in G_i$.

If $X_i = 1$ and $Y_i = 0$ in iteration i , then at least one position of \mathbf{C} gets filled in so $|G_{i+1}| \geq |G_i| + 1$. Therefore the bad event E only occurs if fewer than $\delta \ell_{\text{code}}$ of the X_i take on a 1 or at least one Y_i takes on a 1, giving us:

$$\Pr[E] \leq \Pr \left[\sum_{i=1}^s X_i < \delta \ell_{\text{code}} \right] + \sum_{i=1}^s \Pr[Y_i = 1]$$

For each i , we can bound $\Pr[Y_i = 1] \leq \binom{\lfloor \delta \ell_{\text{code}} \rfloor}{t} / \binom{\ell_{\text{code}}}{t} \leq \delta^t$. If we define $\bar{X} = \frac{1}{s} \sum_{i=1}^s X_i$ we also get:

$$\begin{aligned} \Pr \left[\sum_{i=1}^s X_i < \delta \ell_{\text{code}} \right] &\leq \Pr \left[\bar{X} < 1/p - (1/p - \frac{\delta \ell_{\text{code}}}{s}) \right] \\ &\leq \exp(-2s(1/p - \delta \ell_{\text{code}}/s)^2) \\ &\leq \exp(-s/p) \leq 2^{-\lambda} \end{aligned}$$

where the second inequality follows by the Chernoff-Hoeffding bound. Therefore $\Pr[E] \leq 2^{-\lambda} + s\delta^t = \text{negl}(\lambda)$ which proves the lemma. \square

Use Estimated Success Probability. Instead of looking at the true success probability $\text{Succ}(\tilde{\mathcal{S}}_{\text{fin}})$, which we cannot efficiently compute, let us instead consider an estimated probability $\widetilde{\text{Succ}}(\tilde{\mathcal{S}}_{\text{fin}})$ which is computed in the context of **ExtGame** by sampling $2\lambda(p(\lambda))^2$ different “audit protocol executions” between $\tilde{\mathcal{S}}_{\text{fin}}$ and \mathcal{C}_{fin} and seeing on which fraction of them does $\tilde{\mathcal{S}}$ succeed (while rewinding $\tilde{\mathcal{S}}_{\text{fin}}$ and \mathcal{C}_{fin} after each one). Then, by the Chernoff-Hoeffding bound, we have:

$$\Pr \left[\widetilde{\text{Succ}}(\tilde{\mathcal{S}}_{\text{fin}}) \leq \frac{1}{2p(\lambda)} \mid \text{Succ}(\tilde{\mathcal{S}}_{\text{fin}}) > \frac{1}{p(\lambda)} \right] \leq e^{-\lambda} = \text{negl}(\lambda)$$

Combining the above with (4), we get:

$$\Pr \left[\begin{array}{c} \widetilde{\text{Succ}}(\tilde{\mathcal{S}}_{\text{fin}}) > \frac{1}{2p(\lambda)} \\ \wedge \mathcal{E}^{\tilde{\mathcal{S}}_{\text{fin}}}(\mathcal{C}_{\text{fin}}, 1^\ell, 1^p) = \text{fail}_2 \end{array} \right] > \frac{1}{p'(\lambda)} - \text{negl}(\lambda) \quad (5)$$

Assume Passive Attacker. We now argue that we can replace the active attacker $\tilde{\mathcal{S}}$ with an efficient passive attacker $\hat{\mathcal{S}}$ who always acts as the honest server \mathcal{S} in each protocol execution within the protocol sequence P and the subsequent audit, but can selectively fail by outputting \perp at any point. In particular $\hat{\mathcal{S}}$ just runs a copy of $\tilde{\mathcal{S}}$ and the honest server \mathcal{S} concurrently, and if $\tilde{\mathcal{S}}$ deviates from the execution of \mathcal{S} , it just outputs \perp . Then we claim that, within the context of $\text{ExtGame}_{\hat{\mathcal{S}}, \mathcal{E}}$, we have:

$$\Pr \left[\begin{array}{c} \widetilde{\text{Succ}}(\hat{\mathcal{S}}_{\text{fin}}) > \frac{1}{2p(\lambda)} \\ \wedge \mathcal{E}^{\hat{\mathcal{S}}_{\text{fin}}}(\mathcal{C}_{\text{fin}}, 1^\ell, 1^p) = \text{fail}_2 \end{array} \right] > \frac{1}{p'(\lambda)} - \text{negl}(\lambda) \quad (6)$$

The above probability is equivalent for $\hat{\mathcal{S}}$ and $\tilde{\mathcal{S}}$, up to the latter deviating from the protocol execution without being detected by the client, either during the protocol execution of P or during one of the polynomially many executions of the next read used to compute $\widetilde{\text{Succ}}(\tilde{\mathcal{S}})$ and $\mathcal{E}^{\tilde{\mathcal{S}}}$. The probability that this occurs is negligible, by *authenticity* of ORAM.

Permuted Extractor. We now aim to derive a contradiction from (6). Intuitively, if fail_2 occurs (but fail_1 does not), it means that there is some codeword block \mathbf{c}_j such that $\hat{\mathcal{S}}_{\text{fin}}$ is significantly likelier to fail on a next-read query for which at least one location falls inside \mathbf{c}_j , than it is for a “random” read query. This would imply an attack on next-read pattern hiding. We now make this intuition formal. Consider a modified “permuted extractor” $\mathcal{E}_{\text{perm}}$ who works just like \mathcal{E} with the exception that it permutes the locations used in the **ORed** executions during the extraction process. In particular $\mathcal{E}_{\text{perm}}$ makes the following modifications to \mathcal{E} :

- At the beginning, $\mathcal{E}_{\text{perm}}$ chooses a random permutation $\pi : [\ell_{\text{code}}] \rightarrow [\ell_{\text{code}}]$.
- During each of the s iterations of the audit protocol, $\mathcal{E}_{\text{perm}}$ chooses t indices $j_1, \dots, j_t \in [\ell_{\text{code}}]$ at random as before, *but* it then runs **ORed**($\pi(j_1), \dots, \pi(j_t)$) on the *permuted* values. If the protocol is accepting the extractor $\mathcal{E}_{\text{perm}}$ still “fills-in” the *original* locations: $\mathbf{C}[j_1], \dots, \mathbf{C}[j_t]$ (since we are only analyzing the event fail_2 we do not care about the values in these locations but only if they are filled in or not).

Now we claim that an execution of **ExtGame** the permuted extractor $\mathcal{E}_{\text{perm}}$ is still likely to result in the failure event fail_2 . This follows from “next-read pattern hiding” which ensures that permuting the locations inside of the **ORed** executions (with rewinding) is indistinguishable.

Lemma 3. *The following holds within $\text{ExtGame}_{\hat{\mathcal{S}}, \mathcal{E}_{\text{perm}}}$:*

$$\Pr \left[\begin{array}{c} \widetilde{\text{Succ}}(\hat{\mathcal{S}}_{\text{fin}}) > \frac{1}{2p(\lambda)} \\ \wedge \mathcal{E}_{\text{perm}}^{\hat{\mathcal{S}}_{\text{fin}}}(\mathcal{C}_{\text{fin}}, 1^\ell, 1^p) = \text{fail}_2 \end{array} \right] > \frac{1}{p'(\lambda)} - \text{negl}(\lambda) \quad (7)$$

Proof of Lemma. Assume that (7) does not hold. Then we claim that there is an adversary \mathcal{A} with non-negligible distinguishing advantage in $\text{NextReadGame}_{\mathcal{A}}^b(\lambda)$ against the ORAM.

The adversary \mathcal{A} runs $\hat{\mathcal{S}}$ who chooses a PoR protocol sequence $P_1 = (op_0, \dots, op_{q_2})$, and \mathcal{A} translates this to the appropriate ORAM protocol sequence, as defined by the PORAM scheme. Then \mathcal{A} chooses its own sequence $P_2 = (rop_1, \dots, rop_{q_2})$ of sufficiently many read operations **ORed**(i_1, \dots, i_t) where $i_1, \dots, i_t \in [\ell_{\text{code}}]$ are random distinct indices. It then passes P_1, P_2 to its challenger and gets back the transcripts of the protocol executions for stages (I) and (II) of the game.

The adversary \mathcal{A} then uses the client communication from the stage (I) transcript to run $\hat{\mathcal{S}}$, getting it into some state $\hat{\mathcal{S}}_{\text{fin}}$. It then uses the stage (II) transcripts, to compute $\mathcal{E}_{\text{perm}}^{\hat{\mathcal{S}}_{\text{fin}}}(\mathcal{C}_{\text{fin}}, 1^\ell, 1^p) \stackrel{?}{=} \text{fail}_2$ and to estimate $\widetilde{\text{Succ}}(\hat{\mathcal{S}}_{\text{fin}})$, without knowing the client state \mathcal{C}_{fin} . It does so just by checking on which executions does $\hat{\mathcal{S}}_{\text{fin}}$ abort with \perp and which it runs to completion (here we use that $\hat{\mathcal{S}}$ is semi-honest and never deviates beyond outputting \perp). Lastly \mathcal{A} outputs 1 iff the emulated extraction $\mathcal{E}_{\text{perm}}^{\hat{\mathcal{S}}_{\text{fin}}}(\mathcal{C}_{\text{fin}}, 1^\ell, 1^p) = \text{fail}_2$ and $\widetilde{\text{Succ}}(\hat{\mathcal{S}}_{\text{fin}}) \geq \frac{1}{2p(\lambda)}$.

Let b be the challenger’s bit in the “next-read pattern hiding game”. If $b = 0$ (not permuted) then \mathcal{A} perfectly emulates the distribution of $\mathcal{E}_{\text{perm}}^{\hat{\mathcal{S}}_{\text{fin}}}(\mathcal{C}_{\text{fin}}, 1^\ell, 1^p) \stackrel{?}{=} \text{fail}_2$ and the estimation of $\widetilde{\text{Succ}}(\hat{\mathcal{S}}_{\text{fin}})$ so, by inequality (6):

$$\Pr[\text{NextReadGame}_{\mathcal{A}}^0(\lambda) = 1] \geq 1/p'(\lambda) - \text{negl}(\lambda).$$

If $b = 1$ (permuted) then \mathcal{A} perfectly emulates the distribution of the permuted extractor $\mathcal{E}_{\text{perm}}^{\hat{\mathcal{S}}_{\text{fin}}}(\mathcal{C}_{\text{fin}}, 1^\ell, 1^p) \stackrel{?}{=} \text{fail}_2$ and the estimation of $\widetilde{\text{Succ}}(\hat{\mathcal{S}}_{\text{fin}})$ since, for the latter, it does not matter whether random reads are permuted or not. Therefore, since (7) is false by assumption, we have

$$\Pr[\text{NextReadGame}_{\mathcal{A}}^1(\lambda) = 1] \leq 1/p'(\lambda) - \mu(\lambda)$$

where $\mu(\lambda)$ is non-negligible. This means that the distinguishing advantage of the passive attacker \mathcal{A} is non-negligible in the next-read pattern hiding game, which proves the lemma. \square

Contradiction. Finally, we present an information-theoretic argument showing that, when using the permuted extractor $\mathcal{E}_{\text{perm}}$, the probability of fail_2 is negligible over the choice of the permutation π . Together with inequality (7), this gives us a contradiction.

Lemma 4. *For any (possibly unbounded) $\tilde{\mathcal{S}}$, we have*

$$\Pr \left[\begin{array}{c} \text{Succ}(\tilde{\mathcal{S}}_{\text{fin}}) > \frac{1}{2p(\lambda)} \\ \wedge \mathcal{E}_{\text{perm}}^{\tilde{\mathcal{S}}_{\text{fin}}}(\mathcal{C}_{\text{fin}}, 1^\ell, 1^p) = \text{fail}_2 \end{array} \right] = \text{negl}(\lambda).$$

Proof of Lemma. Firstly, note that an equivalent way of thinking about $\mathcal{E}_{\text{perm}}$ is to have it issue random (unpermuted) read queries just like \mathcal{E} to recover \mathbf{C} , but then permute the locations of \mathbf{C} via some permutation $\pi : [\ell_{\text{code}}] \rightarrow [\ell_{\text{code}}]$ before testing for the event fail_2 . This is simply because we have the distributional equivalence $(\pi(\text{random}), \text{random}) \equiv (\text{random}, \pi(\text{random}))$, where **random** represents the randomly chosen locations for the audit and π is a random permutation. Now, with this interpretation of $\mathcal{E}_{\text{perm}}$, the event fail_2 occurs only if (I) the unpermuted \mathbf{C} contains more than δ fraction of locations with filled in (non \perp) values

so that fail_1 does not occur, and (II) the permuted version $(\mathbf{c}_1, \dots, \mathbf{c}_L) = \mathbf{C}[\pi(1)], \dots, \mathbf{C}[\pi(\ell_{\text{code}})]$ contains some codeword block \mathbf{c}_j with fewer than k/n fraction of filled in (non \perp) values.

We now show that, conditioned on (I) the probability of (II) is negligible over the random choice of π . Fix some index $j \in [L]$ and let us bound the probability that \mathbf{c}_j is the “bad” codeword block with fewer than k filled in values. Let X_1, X_2, \dots, X_n be random variables where X_i is 1 if $\mathbf{c}_j[i] \neq \perp$ and 0 otherwise. Let $\overline{X} \stackrel{\text{def}}{=} \frac{1}{n} \sum_{i=1}^n X_i$. Then, over the randomness of π , the random variables X_1, \dots, X_n are sampled *without replacement* from a population of ℓ_{code} values (location in \mathbf{C}), at least $\delta \ell_{\text{code}}$ of which are 1 ($\neq \perp$) and the rest are 0 ($= \perp$). Therefore, by Hoeffding’s bound for sampling from finite populations without replacement (See section 6 of [18]), we have:

$$\begin{aligned} \Pr[\mathbf{c}_j \text{ is bad}] &= \Pr[\overline{X} < k/n] = \Pr[\overline{X} < \delta - (\delta - k/n)] \\ &\leq \exp(-2n(\delta - k/n)^2) = \text{negl}(\lambda) \end{aligned}$$

By taking a union-bound over all codeword blocks \mathbf{c}_j , we can bound the probability in equation (7) by $\sum_{j=1}^{\ell/k} \Pr[\mathbf{c}_j \text{ is bad}] \leq \text{negl}(\lambda)$.

We have already shown that fail_1 only occurs with negligible probability. We now showed that fail_2 for the permuted extractor also occurs with negligible probability, while the adversary succeeds with non-negligible probability. \square

Combining the above lemma with equation (7), we get a contradiction, showing that the assumption in equation (1) cannot hold. Thus, as long as the adversary succeeds with non-negligible probability during audits, the extractor will also succeed with non-negligible probability in extracting the whole memory contents correctly.

6 ORAM Instantiation

The notion of ORAM was introduced by Goldreich and Ostrovsky [13], who also introduced the so-called *hierarchical scheme* having the structure seen in Figures 1 and 6.2. Since then several improvements to the hierarchical scheme have been given, including improved rebuild phases and the use of advanced hashing techniques [31, 26, 15].

We examine a particular ORAM scheme of Goodrich and Mitzenmacher [15] and show that (with minor modifications) it satisfies *next-read pattern hiding* security. Therefore, this scheme can be used to instantiate our PORAM construction. We note that most other ORAM schemes from the literature that follow the hierarchical structure also seemingly satisfy next-read pattern hiding, and we only focus on the above example for concreteness. However, in Appendix C, we show that it is *not* the case that *every* ORAM scheme satisfies next-read pattern hiding, and in fact give an example of a contrived scheme which does not satisfy this notion and makes our construction of PORAM completely insecure. We also believe that there are natural schemes, such as the ORAM of Shi et al. [28], which do not satisfy this notion. Therefore, next-read pattern hiding is a meaningful property beyond standard ORAM security and must be examined carefully.

Overview. We note that ORAM schemes are generally not described as protocols, but simply as a data structure in which the client’s encrypted data is stored on the server. Each time that a client wants to perform a read or write to some address i of her memory, this operation is translated into a series of read/write operations on this data structure inside the server’s storage. In other words, the (honest) server does not perform any computation at all during these ‘protocols’, but simply allows the client to access arbitrary locations inside this data structure.

Most ORAM schemes, including the one we will use below, follow a *hierarchical structure*. They maintain several *levels* of hash tables on the server, each holding encrypted address-value pairs, with lower tables

having higher capacity. The tables are managed so that the most recently accessed data is kept in the top tables and the least recently used data is kept in the bottom tables. Over time, infrequently accessed data is moved into lower tables (obviously).

To write a value to some address, just insert the encrypted address-value pair in the top table. To read the value at some address, one hashes the address and checks the appropriate position in the top table. If it *is* found in that table, then one hides this fact by sequentially checking random positions in the remaining tables. If it *is not* found in the top table, then one hashes the address again and checks the second level table, continuing down the list until it is found, and then accessing random positions in the remaining tables. Once all of the tables have been accessed, the found data is written into the top table. To prevent tables from overflowing (due to too many item insertions), there are additional periodic *rebuild phases* which obviously moves data from the smaller tables to larger tables further down.

Security Intuition. The reason that we always write found data into the top table after any read, is to protect the privacy of repeatedly reading the same address, and ensuring that this looks the same as reading various different addresses. In particular, reading the same address twice will not need to access the same locations on the server, since after the first read, the data will already reside in the top table, and the random locations will be read at lower tables.

At any point in time, after the server observes many read/write executions, any subsequent read operation just accesses completely random locations in each table, from the point of view of the server. This is the main observation needed to argue standard pattern hiding. For next-read pattern hiding, we notice that we can extend the above to any set of q distinct executions of a subsequent read operation with distinct addresses (each execution starting in the same client/server state). In particular, each of the q operations just accesses completely random locations in each table, independently of the other operations, from the point of view of the server.

One subtlety comes up when the addresses are not completely *distinct* from each other, as is the case in our definition where each address can appear in multiple separate multi-read operations. The issue is that doing a read operation on the same address twice with rewinding will reveal the level at which the data for that address is stored, thus revealing some information about which address is being accessed. One can simply observe at which level do the accesses begin to differ in the two executions. We fix this issue by modifying a scheme so that, instead of accessing freshly chosen random positions in lower tables once the correct value is found, we instead access *pseudorandom positions that are determined by the address being read and the operation count*. That way, any two executions which read the same address *starting from the same client state* are *exactly* the same and do not reveal anything beyond this. Note that, without state rewinds, this still provides regular pattern hiding.

6.1 Technical Tools

Our construction uses the standard notion of a *pseudorandom-function* (PRF) where $F(K, x)$ denote the evaluation of the PRF F on input x with key K . We also rely on a *symmetric-key encryption* scheme secure against *chosen-plaintext attacks*, and let $\text{Enc}(K, \cdot), \text{Dec}(K, \cdot)$ denote the encryption/decryption algorithms with key K .

Encrypted cuckoo table. An encrypted cuckoo table [25, 20] consists of three arrays (T_1, T_2, S) that hold ciphertexts of some fixed length. The arrays T_1 and T_2 are both of size m and serve as *cuckoo-hash tables* while S is an array of size s and serves as an auxiliary *stash*. The data structure uses two hash functions $h_1, h_2 : [\ell] \rightarrow [m]$. Initially, all entries of the arrays are populated with independent encryptions of a special symbol \perp . To retrieve a ciphertext associated with an address i , one decrypts all of the ciphertexts in S , as well as the ciphertexts at $T_1[h_1[i]]$ and $T_2[h_2[i]]$ (thus at most $s + 2$ decryptions are performed). If any of these ciphertexts decrypts to a value of the form (i, v) , then v is the returned output. To insert an address-value pair (i, v) , encrypt it and write the ciphertext ct to position $T_1[h_1(i)]$, retrieving whatever

ciphertext ct_1 was there before. If the original ciphertext ct_1 decrypts to \perp , then stop. Otherwise, if ct_1 decrypts to a pair (j, w) , then re-encrypt the pair and write the resulting ciphertext to $T_2[h_2(j)]$, again retrieving whatever ciphertext ct_2 was there before. If ct_2 decrypts to \perp , then stop, and otherwise continue this process iteratively with ciphertexts $\text{ct}_3, \text{ct}_4, \dots$. If this process continues for $t = c \log n$ steps, then ‘give up’ and just put the last evicted ciphertext ct_t into the first available spot in the stash S . If S is full, then the data structure fails.

We will use the following result sketched in [15]: If $m = (1 + \varepsilon)n$ for some constant $\varepsilon > 0$, and h_1, h_2 are random functions, then after n items are inserted, the probability that S has k or more items written into it is $O(1/n^{k+2})$. Thus, if S has at least λ slots, then the probability of a failure after n insertions is negligible in λ .

Oblivious table rebuilds. We will assume an oblivious protocol for the following task. At the start of the protocol, the server holds encrypted cuckoo hash tables C_1, \dots, C_r . The client has two hash functions h_1, h_2 . After the oblivious interaction, the server holds a new cuckoo hash table C'_r that results from decrypting the data in C_1, \dots, C_r , deleting data for duplicated locations with preference given to the copy of the data in the lowest index table, encrypting each index-value pair again, and then inserting the ciphertexts into C'_r using h_1, h_2 .

Implementing this task efficiently and obliviously is an intricate task. See [15] and [26] for different methods, which adapt the usage of oblivious sorting first introduced in [13].

6.2 ORAM Scheme

We can now describe the scheme of Goodrich et al, with our modifications for next-read pattern hiding. As ingredients, this scheme will use a PRF F and an encryption scheme (Enc, Dec) . A visualization of the server’s data structures is given in Figure 6.2.

Onit $(1^\lambda, 1^w, \ell)$: Let L the smallest integer such that $2^L > \ell$. The client chooses $2L$ random keys $K_{1,1}, K_{1,2}, \dots, K_{L,1}, K_{L,2}$ and $2L$ additional random keys $R_{1,1}, R_{1,2}, \dots, R_{L,1}, R_{L,2}$ to be used for pseudo-random functions, and initializes a counter ctr to 0. It also selects an encryption key for the IND-CPA secure scheme. It instructs the server to allocate the following data structures:

- An empty array A_0 that will change size as it is used.
- L empty cuckoo hash tables C_1, \dots, C_L where the parameters in C_j are adjusted to hold 2^j data items with a negligible (in λ) probability of overflow when used with random hash functions.

The client state consists of all of the keys $(K_{j,0}, K_{j,1})_{j \in [L]}$, $(R_{j,0}, R_{j,1})_{j \in [L]}$, the encryption key, and ctr .

ORead (i_1, \dots, i_t) : The client starts by initializing an array found of t flags to **false**. For each index i_j to be read, the client does the following. For each level $k = 1, \dots, L$, the client executes

- Let $C_k = (T_1^{(k)}, T_2^{(k)}, S^{(k)})$
- If $\text{found}[j] = \text{false}$, read and decrypt all of $S^{(k)}$, $T_1^{(k)}[F(K_{k,1}, i_j)]$ and $T_2^{(k)}[F(K_{k,2}, i_j)]$. If the data is in any of these slots, set $\text{found}[j]$ to **true** and remember the value as v_j .
- Else, if $\text{found}[j] = \text{true}$, then instead read all of $S^{(k)}$, $T_1^{(k)}[F(R_{k,1}, i_j \parallel \text{ctr})]$ and $T_2^{(k)}[F(R_{k,2}, i_j \parallel \text{ctr})]$ and ignore the results. Note that the counter value is used to create random reads when the state is not reset, while providing the same random values if the state is reset.

Finally, it encrypts and appends (i_j, v_j) to the end of A_0 and continues to the next index i_{j+1} . We note that above, when accessing a table using the output of F , we are interpreting the bit string output by F as a random index from the appropriate range.

After all the indices have been read and written to A_0 , the client initiates a *rebuild phase*, the description of which we defer for now.

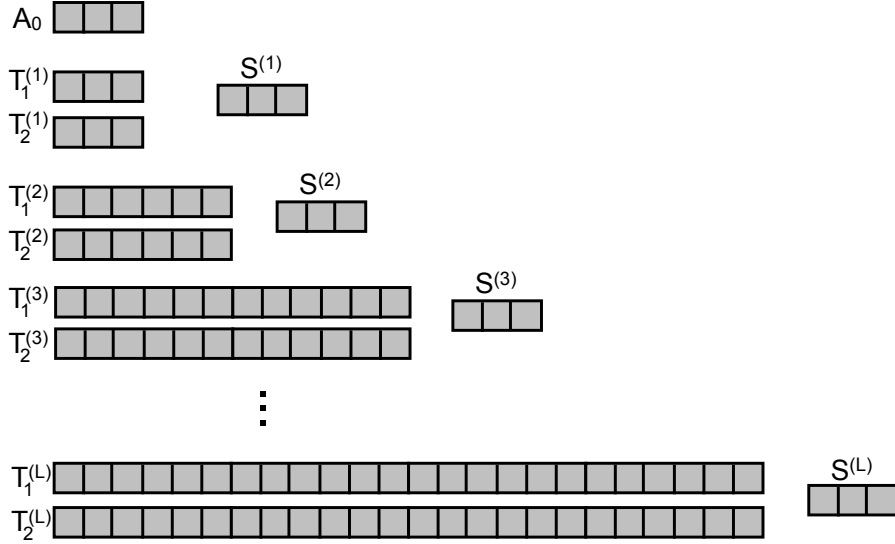


Figure 2: Server data structures in the ORAM instantiation.

Rebuild phase. We complete the scheme description by describing a rebuild phase, which works as follows. The client repeats the following process until A_0 is empty:

- Increment ctr .
- Remove and decrypt an item from A_0 , calling the result (j, v) .
- Let $r \geq 0$ be the largest integer such that 2^r divides $(\text{ctr} \bmod 2^L)$.
- Select new keys $K_{r,1}, K_{r,2}$ and use the functions $F(K_{r,1}, \cdot)$ and $F(K_{r,2}, \cdot)$ as h_1 and h_2 to obviously build a new cuckoo table C'_r holding the removed item (j, v) and all of the data items in C_1, \dots, C_{r-1} , freshly re-encrypted and with duplicates removed.
- Then, for $j = 1$ to $r - 1$, set $K_{j,1}, K_{j,2}$ to fresh random keys and set the cuckoo tables C_1, \dots, C_r to be new, empty tables and C_r to be C'_r .

Note that the remaining tables C_{r+1}, \dots, C_L are not touched.

We can implement the rebuild phase using the any of the protocols (with small variations) from [15, 16]. The most efficient gives an amortized overhead of $\log \ell$ operations for all rebuilds, assuming that the client can *temporarily* locally store ℓ^δ memory slots during the protocol (but the client does need to store them between executions of the protocol). If we only allow the client to store a constant number of slots at any one time, then the we incur an overhead of $\log^2 \ell$. In either case the worst-case overhead is $O(\ell)$. Using the de-amortization techniques from [16, 24], we can achieve worst-case complexity of $\log^2 \ell$, at the cost of doubling the server storage. This technique was analyzed in the original ORAM security setting, but it is not hard to extend our proof to show that it preserves next-read pattern hiding as well.

6.3 Next-Read Pattern Hiding

Theorem 2. *Assuming that F is a secure PRF, and the underlying encryption scheme is chosen-plaintext secure, then the scheme **O** described above is next-read pattern hiding.*

Proof. We show that for any efficient adversary \mathcal{A} , the probabilities that \mathcal{A} outputs 1 when playing either $\text{NextReadGame}_{\mathcal{A}}^0$ or $\text{NextReadGame}_{\mathcal{A}}^1$ differs by only a negligible amount. In these games, the adversary \mathcal{A}

provides two tuples of operations $P_1 = (op_1, \dots, op_{q_1})$ and $P_2 = (rop_1, \dots, rop_{q_2})$, the latter being all multi-reads, and a permutation π on $[\ell]$. Then in $\text{NextReadGame}_{\mathcal{A}}^0$, \mathcal{A} is given the transcript of an honest client and server executing P_1 , as well as the transcript of executing the multi-reads in P_2 with rewinds after each operation, while in $\text{NextReadGame}_{\mathcal{A}}^1$ it is given the same transcript except that second part is generated by first permuting the addresses in P_2 according to π .

We need to argue that these inputs are computationally indistinguishable. For our analysis below, we assume that a rebuild phase never fails, as this event happens with negligible probability in λ , as discussed before. We start by modifying the execution of the games in two ways that are shown to be undetectable by \mathcal{A} . The first change will show that all of the accesses into tables appear to the adversary to be generated by random functions, and the second change will show that the ciphertexts do not reveal any usable information for the adversary.

First, whenever keys $K_{j,1}, K_{j,2}$ are chosen and used with the function F , we use random functions $g_{j,1}, g_{j,2}$ in place of $F(K_{j,1}, \cdot)$ and $F(K_{j,2}, \cdot)$.¹¹ We do the same for the $R_{j,1}, R_{j,2}$ keys, calling the random functions $r_{j,1}$ and $r_{j,2}$. This change only changes the behavior of \mathcal{A} by a negligible amount, as otherwise we could build a distinguisher to contradict the PRF security of F via a standard hybrid argument over all of the keys chosen during the game.

The second change we make is that all of the ciphertexts in the transcript are replaced with independent encryptions of equal-length strings of zeros. We claim that this only affects the output distribution of \mathcal{A} by a negligible amount, as otherwise we could build an adversary to contradict the IND-CPA security of the underlying encryption scheme via a standard reduction. Here it is crucial that, after each rewind, the client chooses new randomness for the encryption scheme.

We now complete the proof by showing that the distribution of the transcripts given to \mathcal{A} is identical in the modified versions of $\text{NextReadGame}_{\mathcal{A}}^0$ and $\text{NextReadGame}_{\mathcal{A}}^1$. To see why this is true, let us examine what is in one of the game transcripts given to \mathcal{A} . The transcript for the execution of P_1 consists of **ORead** and **OWrite** transcripts, which are accesses to indices in the cuckoo hash tables, ciphertext writes into A_0 , and rebuild phases. Finally the execution of P_2 (either permuted by π or not) with rewinds generates a transcript that consists of several accesses to the cuckoo hash tables, each followed by writes to A_0 and a rebuild phase.

By construction of the protocol, in the modified game the only part of the transcript that depends on the addresses in P_2 are the reads into $T_1^{(k)}$ and $T_2^{(k)}$ for each k . All other parts of the transcript are oblivious scans of the $S^{(k)}$ arrays and oblivious table rebuilds which do not depend on the addresses (recall the ciphertexts in these transcripts are encryptions of zeros). Thus we focus on the indices read in each $T_1^{(k)}$ and $T_2^{(k)}$, and need to show that, in the modified games, the distribution of these indices does not depend on the addresses in P_2 .

The key observation is that, after the execution of P_1 , the state of the client is such that each address i will induce a uniformly random sequence of indices in the tables that is independent of the indices read for any other address and independent of the transcript for P_1 . If the data is in the cuckoo table at level k , then the indices will be

$$(g_{j,1}(i))_{j=1}^k \text{ and } (r_{j,1}(i \parallel \text{ctr}))_{j=k+1}^L.$$

Thus each i induces a random sequence, and each address will generate an independent sequence. We claim moreover that the sequence for i is independent of the transcript for P_1 . This follows from the construction: For the indices derived from $r_{j,1}$ and $r_{j,2}$, the transcript for P_1 would have always used a lower value for ctr . For the indices derived from $g_{j,1}$ and $g_{j,2}$, we have that the execution of P_1 would not have evaluated those functions on input i : If i was read during P_1 , then i would have been written to A_0 and a rebuild phase would have chosen new random functions for $g_{j,1}$ and $g_{j,2}$ before the address/value pair i was placed in the

¹¹As usual, instead of actually picking and using a random function, which is an exponential task, we create random numbers whenever necessary, and remember them. Since there will be only polynomially-many interactions, this only requires polynomial time and space.

j -th level table again.

With this observation we can complete the proof. When the modified games are generating the transcript for the multi-read operations in P_2 , each individual read for an index i induces an random sequence of table reads among its other oblivious operations. But since each i induces a completely random sequence and permuting the addresses will only permute the random sequences associated with the addresses, the distribution of the transcript is unchanged. Thus no adversary can distinguish these games, which means that no adversary could distinguish NextReadGame_A^0 and NextReadGame_A^1 , as required. \square

6.4 Authenticity, Extensions & Optimizations

Authenticity. To achieve authenticity we sketch how to employ the technique introduced in [13]. A straightforward attempt is to tag every ciphertext stored on the server along with its location on the server using a message authentication code (MAC). But this fails because the sever can “roll back” changes to the data by replacing ciphertexts with previously stored ones at the same location. We can generically fix this by using the techniques of *memory checking* [5, 23, 11] at some additional logarithmic overhead. However, it also turns out that authenticity can also be added at almost no cost to several specific constructions, as we describe below.

Goldreich and Ostrovsky showed that any ORAM protocol supporting *time labeled simulation* (TLS) can be modified to achieve authenticity without much additional complexity. We say that an ORAM protocol *supports TLS* if there exists an efficient algorithm Q such that, after the j -th message is sent to the server, for each index x on the server memory, the number of times x has been written to is equal to $Q(j, x)$.¹² Overall, one implements the above tagging strategy, and also includes $Q(j, x)$ with the data being tagged, and when reading one recomputes $Q(j, x)$ to verify the tag.

Our scheme can be shown to support TLS in a manner very similar to the original hierarchical scheme [13]. The essential observation, also used there, is that the table indices are only written to during a rebuild phase, so by tracking the number of executed rebuild phases we can compute how many times each index of the table was written to.

Extensions and optimizations. The scheme above is presented in a simplified form that can be made more efficient in several ways while maintaining security.

- The keys in the client state can be derived from a single key by appropriately using the PRF. This shrinks the client state to a single key and counter.
- The initial table C_1 can be made larger to reduce the number of rebuild phases (although this does not affect the asymptotic complexity).
- We can collapse the individual oblivious table rebuilds into one larger rebuild.
- It was shown in [17] that all of the L cuckoo hash tables can share a single $O(\lambda)$ -size stash S while still maintaining a negligible chance of table failure.
- Instead of doing table rebuilds all at once, we can employ a technique that allows for them to be done incrementally, allowing us to achieve worst-case rather than amortized complexity guarantees [16, 24]. These techniques come at the cost of doubling the server storage.
- The accesses to cuckoo tables on each level during a multi-read can be done in parallel, which reduces the round complexity of that part to be independent of t , the number of addresses being read.

We can also extend this scheme to support a dynamically changing memory size. This is done by simply allocating different sized tables during a rebuild that eliminate the lower larger tables or add new ones of the appropriate size. This modification will achieve next-read pattern hiding security, but it will not be standard pattern-hiding secure, as it leaks some information about the number of memory slots in use. One

¹²Here we mean actual writes on the server, and not **O**Write executions.

can formalize this, however, in a pattern-hiding model where any two sequences with equal memory usage are required to be indistinguishable.

Efficiency. In this scheme the client stores the counter and the keys, which can be derived from a single key using the PRF. The server stores $\log \ell$ tables, where the j -th table requires $2^j + \lambda$ memory slots, which sums to $O(\ell + \lambda \cdot \log \ell)$. Using the optimization above, we only need a single stash, reducing the sum to $O(\ell + \lambda)$. When executing **ORed**, each index read requires accessing two slots plus the λ stash slots in each of the $\log \ell$ tables, followed by a rebuild. **OWrite** is simply one write followed by a rebuild phase. The table below summarizes the efficiency measures of the scheme.

Client Storage	$O(1)$
Server Storage	$O(\ell + \lambda)$
Read Complexity	$O(\lambda \cdot \log \ell) + \text{RP}$
Write Complexity	$O(1) + \text{RP}$

Table 1: Efficiency of ORAM scheme above. “RP” denotes the aggregate cost of the rebuild phases, which is $O(\log \ell)$, or $O(\log^2 \ell)$ in the worst-case, per our discussion above.

7 Efficiency

We now look at the efficiency of our PORAM construction, when instantiated with the ORAM scheme from section 6 (we assume the rebuild phases are implemented via the Goodrich-Mitzemacher algorithm [15] with the worst-case complexity optimization [16, 24].) Since our PORAM scheme preserves (standard) ORAM security, we analyze its efficiency in two ways. Firstly, we look at the overhead of PORAM scheme on top of just storing the data inside of the ORAM without attempting to achieve any PoR security (e.g., not using any error-correcting code etc.). Secondly, we look at the overall efficiency of PORAM. Third, we compare it with dynamic PDP [12, 30] which does not employ erasure codes and does not provide full retrievability guarantee. In the table below, ℓ denotes the size of the client data and λ is the security parameter. We assume that the ORAM scheme uses a PRF whose computation takes $O(\lambda)$ work.

<i>PORAM Efficiency</i>	vs. ORAM	Overall	vs. Dynamic PDP [12]
Client Storage	Same	$O(\lambda)$	Same
Server Storage	$\times O(1)$	$O(\ell)$	$\times O(1)$
Read Complexity	$\times O(1)$	$O(\lambda \log^2 \ell)$	$\times O(\log \ell)$
Write Complexity	$\times O(\lambda)$	$O(\lambda^2 \times \log^2 \ell)$	$\times O(\lambda \times \log \ell)$
Audit Complexity	Read $\times O(\lambda)$	$O(\lambda^2 \times \log^2 \ell)$	$\times O(\log \ell)$

By modifying the underlying ORAM to dynamically resize tables during rebuilds, the resulting PORAM instantiation will achieve the same efficiency measures as above, but with ℓ taken to be amount of memory currently used by the memory access sequence. This is in contrast to the usual ORAM setting where ℓ is taken to be a (perhaps large) upper bound on the total amount of memory that will ever be used.

Acknowledgements

Alptekin Küpgü would like to acknowledge the support of TÜBİTAK, the Scientific and Technological Research Council of Turkey, under project number 112E115. David Cash and Daniel Wichs are sponsored by DARPA under agreement number FA8750-11-C-0096. The U.S. Government is authorized to reproduce

and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views expressed are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. Distribution Statement “A” (Approved for Public Release, Distribution Unlimited).

References

- [1] G. Ateniese, R. C. Burns, R. Curtmola, J. Herring, L. Kissner, Z. N. J. Peterson, and D. Song. Provable data possession at untrusted stores. In P. Ning, S. D. C. di Vimercati, and P. F. Syverson, editors, *ACM CCS 07*, pages 598–609. ACM Press, Oct. 2007.
- [2] G. Ateniese, S. Kamara, and J. Katz. Proofs of storage from homomorphic identification protocols. In M. Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 319–333. Springer, Dec. 2009.
- [3] G. Ateniese, R. D. Pietro, L. V. Mancini, and G. Tsudik. Scalable and efficient provable data possession. Cryptology ePrint Archive, Report 2008/114, 2008. <http://eprint.iacr.org/>.
- [4] M. Bellare and O. Goldreich. On defining proofs of knowledge. In E. F. Brickell, editor, *CRYPTO’92*, volume 740 of *LNCS*, pages 390–420. Springer, Aug. 1993.
- [5] M. Blum, W. S. Evans, P. Gemmell, S. Kannan, and M. Naor. Checking the correctness of memories. *Algorithmica*, 12(2/3):225–244, 1994.
- [6] K. D. Bowers, A. Juels, and A. Oprea. HAIL: a high-availability and integrity layer for cloud storage. In E. Al-Shaer, S. Jha, and A. D. Keromytis, editors, *ACM CCS 09*, pages 187–198. ACM Press, Nov. 2009.
- [7] K. D. Bowers, A. Juels, and A. Oprea. Proofs of retrievability: theory and implementation. In R. Sion and D. Song, editors, *CCSW*, pages 43–54. ACM, 2009.
- [8] B. Chen, R. Curtmola, G. Ateniese, and R. C. Burns. Remote data checking for network coding-based distributed storage systems. In A. Perrig and R. Sion, editors, *CCSW*, pages 31–42. ACM, 2010.
- [9] R. Curtmola, O. Khan, R. Burns, and G. Ateniese. Mr-pdp: Multiple-replica provable data possession. In *ICDCS*, 2008.
- [10] Y. Dodis, S. P. Vadhan, and D. Wichs. Proofs of retrievability via hardness amplification. In O. Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 109–127. Springer, Mar. 2009.
- [11] C. Dwork, M. Naor, G. N. Rothblum, and V. Vaikuntanathan. How efficient can memory checking be? In O. Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 503–520. Springer, Mar. 2009.
- [12] C. C. Erway, A. Küpçü, C. Papamanthou, and R. Tamassia. Dynamic provable data possession. In E. Al-Shaer, S. Jha, and A. D. Keromytis, editors, *ACM CCS 09*, pages 213–222. ACM Press, Nov. 2009.
- [13] O. Goldreich and R. Ostrovsky. Software protection and simulation on oblivious RAMs. *Journal of the ACM*, 43(3):431–473, 1996.
- [14] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.
- [15] M. T. Goodrich and M. Mitzenmacher. Privacy-preserving access of outsourced data via oblivious RAM simulation. In L. Aceto, M. Henzinger, and J. Sgall, editors, *ICALP 2011, Part II*, volume 6756 of *LNCS*, pages 576–587. Springer, July 2011.

- [16] M. T. Goodrich, M. Mitzenmacher, O. Ohrimenko, and R. Tamassia. Oblivious RAM simulation with efficient worst-case access overhead. In *CCSW*, pages 95–100, 2011.
- [17] M. T. Goodrich, M. Mitzenmacher, O. Ohrimenko, and R. Tamassia. Privacy-preserving group data access via stateless oblivious ram simulation. In *SODA*, pages 157–167, 2012.
- [18] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.
- [19] A. Juels and B. S. Kaliski Jr. Pors: proofs of retrievability for large files. In P. Ning, S. D. C. di Vimercati, and P. F. Syverson, editors, *ACM CCS 07*, pages 584–597. ACM Press, Oct. 2007.
- [20] A. Kirsch, M. Mitzenmacher, and U. Wieder. More robust hashing: Cuckoo hashing with a stash. *SIAM J. Comput.*, 39(4):1543–1561, 2009.
- [21] A. Küpçü. *Efficient Cryptography for the Next Generation Secure Cloud*. PhD thesis, Brown University, 2010.
- [22] A. Küpçü. *Efficient Cryptography for the Next Generation Secure Cloud: Protocols, Proofs, and Implementation*. Lambert Academic Publishing, 2010.
- [23] M. Naor and G. N. Rothblum. The complexity of online memory checking. In *46th FOCS*, pages 573–584. IEEE Computer Society Press, Oct. 2005.
- [24] R. Ostrovsky and V. Shoup. Private information storage (extended abstract). In *29th ACM STOC*, pages 294–303. ACM Press, May 1997.
- [25] R. Pagh and F. F. Rodler. Cuckoo hashing. *J. Algorithms*, 51(2):122–144, 2004.
- [26] B. Pinkas and T. Reinman. Oblivious RAM revisited. In T. Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 502–519. Springer, Aug. 2010.
- [27] H. Shacham and B. Waters. Compact proofs of retrievability. In J. Pieprzyk, editor, *ASIACRYPT 2008*, volume 5350 of *LNCS*, pages 90–107. Springer, Dec. 2008.
- [28] E. Shi, T.-H. H. Chan, E. Stefanov, and M. Li. Oblivious ram with $o((\log n)^3)$ worst-case cost. In D. H. Lee and X. Wang, editors, *ASIACRYPT*, volume 7073 of *Lecture Notes in Computer Science*, pages 197–214. Springer, 2011.
- [29] E. Stefanov, M. van Dijk, A. Oprea, and A. Juels. Iris: A scalable cloud file system with efficient integrity checks. Cryptology ePrint Archive, Report 2011/585, 2011. <http://eprint.iacr.org/>.
- [30] Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou. Enabling public verifiability and data dynamics for storage security in cloud computing. In M. Backes and P. Ning, editors, *ESORICS 2009*, volume 5789 of *LNCS*, pages 355–370. Springer, Sept. 2009.
- [31] P. Williams, R. Sion, and B. Carbunar. Building castles out of mud: practical access pattern privacy and correctness on untrusted storage. In P. Ning, P. F. Syverson, and S. Jha, editors, *ACM CCS 08*, pages 139–148. ACM Press, Oct. 2008.

A Simple Dynamic PoR with Square-Root Complexity

We sketch a very simple construction of dynamic PoR that achieves *sub-linear* complexity in its read, write and audit operations. Although the scheme is asymptotically significantly worse than our PORAM solution as described in the main body, it is significantly simpler and may be of interest for some practical parameter settings.

The construction starts with the first dynamic PoR proposal from the introduction. To store a memory $\mathbf{M} \in \Sigma^\ell$ on the server, the client divides it into $L = \sqrt{\ell}$ consecutive message blocks $(\mathbf{m}_1, \dots, \mathbf{m}_L)$, each containing $L = \sqrt{\ell}$ symbols. The client then encodes each of the message blocks \mathbf{m}_i using an $(n = 2L, k = L, d = L + 1)$ -erasure code (e.g., Reed-Solomon tolerating L erasures), to form a codeword block \mathbf{c}_i , and concatenates the codeword blocks to form a string $\mathbf{C} = (\mathbf{c}_1, \dots, \mathbf{c}_L) \in \Sigma^{2\ell}$ which it then stores on the server. We can assume the code is systematic so that the message block \mathbf{m}_i resides in the first L symbols of the corresponding codeword block \mathbf{c}_i . In addition, the client initializes a *memory checking scheme* [5, 23, 11], which it uses to authenticate each of the 2ℓ codeword symbols within \mathbf{C} .

To read a location $j \in [\ell]$ of memory, the client computes the index $i \in [L]$ of the message block \mathbf{m}_i containing that location, and downloads the appropriate symbol of the codeword block \mathbf{c}_i which contains the value $\mathbf{M}[j]$ (here we use that the code is systematic), which it checks for authenticity via the memory checking scheme. To write to a location $j \in [\ell]$ the client downloads the entire corresponding codeword block \mathbf{c}_i (checking for authenticity) decodes \mathbf{m}_i , changes the appropriate location to get an updated block \mathbf{m}'_i and finally re-encodes it to get \mathbf{c}'_i which it then writes to the server, updating the appropriate authentication information within the memory checking scheme. The audit protocol selects $t = \lambda$ (security parameter) random positions within *every* codeword block \mathbf{c}_i and checks them for authenticity via the memory checking scheme.

The read and write protocols of this scheme each execute the memory checking read protocol to read and write 1 and $\sqrt{\ell}$ symbols respectively. The audit protocol reads and checks $\lambda\sqrt{\ell}$ symbols. Assuming an efficient (poly-logarithmic) memory checking protocol, this means actual complexity of these protocols incurs another $O(\log \ell)$ factor and another constant factor increase in server storage. Therefore the complexity of the reads, writes, and audit is $O(1), O(\sqrt{\ell}), O(\sqrt{\ell})$ respectively, ignoring factors that depend on the security parameter or are polylogarithmic in ℓ .

Note that the above scheme actually gives us a natural trade-off between the complexity of the writes and the audit protocol. In particular, for any $\delta > 0$, we can set the message block size to $L_1 = \ell^\delta$ symbols, so that the client memory \mathbf{M} now consists of $L_2 = \ell^{1-\delta}$ such blocks. In this case, the complexity of reads, writes, and audits becomes $O(1), O(\ell^\delta), O(\ell^{1-\delta})$ respectively.

B Standard Pattern Hiding for ORAM

We recall an equivalent definition to the one introduced by Goldreich and Ostrovsky [13]. Informally, standard pattern hiding says that an (arbitrarily malicious and efficient) adversary cannot detect which sequence of instructions a client is executing via the ORAM protocols.

Formally, for a bit b and an adversary \mathcal{A} , we define the game $\text{ORAMGame}_{\mathcal{A}}^b(\lambda)$ as follows:

- The attacker $\mathcal{A}(1^\lambda)$ outputs two equal-length ORAM protocol sequences $Q_0 = (op_0, \dots, op_q), Q_1 = (op'_0, \dots, op'_q)$. We require that for each index j , the operations op_j and op'_j only differ in the location they access and the values they are writing, but otherwise correspond to the same operation (read or write).
- The challenger initializes an honest client \mathcal{C} and server \mathcal{S} , and sequentially executes the operations in Q_b , between \mathcal{C} and \mathcal{S} .
- Finally, \mathcal{A} is given the complete transcript of all the protocol executions, and he outputs a bit \tilde{b} , which is the output of the game.

We say that an ORAM protocol is pattern hiding if for all efficient adversaries \mathcal{A} we have:

$$|\Pr[\text{ORAMGame}_{\mathcal{A}}^0(\lambda) = 1] - \Pr[\text{ORAMGame}_{\mathcal{A}}^1(\lambda) = 1]| \leq \text{negl}(\lambda).$$

Sometimes we also want to achieve a stronger notion of security where we also wish to hide whether each operation is a read or a write. This can be done generically by always first executing a read for the desired location and then executing a write to either just write-back the read value (when we only wanted to do a read) or writing in a new value.

C Standard ORAM Security Does not Suffice for PORAM

In this section we construct an ORAM that *is* secure in the usual sense but *is not* next-read pattern hiding. In fact, we will show something stronger: If the ORAM below were used to instantiate our PORAM scheme then the resulting dynamic PoR scheme is not secure. This shows that some notion of security beyond regular ORAM is necessary for the security PORAM.

Counterexample construction. We can take any ORAM scheme (e.g., the one in Section 6 for concreteness) and modify it by “packing” multiple consecutive logical addresses into a single slot of the ORAM. In particular, if the client initializes the modified ORAM (called MORAM within this section) with alphabet $\Sigma = \{0,1\}^w$, it will translate this into initializing the original ORAM with the alphabet $\Sigma^n = \{0,1\}^{nw}$, where each symbol in the modified alphabet “packs” together n symbols of the original alphabet. Assume this is the same n as the codeword length in our PORAM protocol.

Whenever the client wants to read some address i using MORAM, the modified scheme looks up where it was packed by computing $j = \lfloor i/n \rfloor$, uses the original ORAM scheme to execute $\mathbf{ORed}(j)$, and then parses the resulting output as $(v_0, \dots, v_{n-1}) \in \Sigma^n$, and returns $v_{i \bmod n}$. To write v to address i , MORAM runs ORAM scheme’s $\mathbf{ORed}(\lfloor i/n \rfloor)$ to get (v_0, \dots, v_{n-1}) as before, then sets $v_{i \bmod n} \leftarrow v$ and writes the data back via ORAM scheme’s $\mathbf{OWrite}(\lfloor i/n \rfloor, (v_0, \dots, v_{n-1}))$. It is not hard to show that this modified scheme retains standard ORAM security, since it hides which locations are being read/written.

We next discuss why this modification causes the MORAM to not be NRPH secure. Consider what happens if the client issues a read for an address, say $i = 0$, and then is rewound and reads another address that was packed into the same ORAM slot, say $i + 1$. Both operations will cause the client to issue $\mathbf{ORed}(0)$. And since our MORAM was deterministic, the client will access exactly same table indices at every level on the server on both runs. But, if these addresses were permuted to not be packed together (e.g., blocks were packed using equivalence classes of their indices $(\bmod \ell/n)$), then the client will issue \mathbf{ORed} commands on different addresses, reading different table positions (with high probability), thus allowing the server to distinguish which case it was in and break NRPH security.

This establishes that the modified scheme is not NRPH secure. To see why PORAM is not secure with MORAM, consider an adversary that, after a sequence of many read/write operations, randomly deletes one block of its storage (say, from the lowest level cuckoo table). If this block happens to contain a non-dummy ciphertext that contains actual data (which occurs with reasonable probability), then this attack corresponds to deleting some codeword block in full (because all codeword blocks corresponding to a message block was packed in the same ORAM storage location), even though the server does not necessarily know which one. Therefore, the underlying message block can never be recovered from the attacker. But this adversary can still pass an audit with good probability, because the audit would only catch the adversary if it happened to access the deleted block during its reads either by (1) selecting exactly this location to check during the audit, (2) reading this location in the cuckoo table slot as a dummy read. This happens with relatively low probability, around $1/\ell$, where ℓ is the number of addresses in the client memory.

To provide some more intuition, we can also examine why this same attack (deleting a random location in the lowest level cuckoo table) does not break PORAM when instantiated with the ORAM implementation

from Section 6 that is NRPH secure. After this attack, the adversary still maintains a good probability of passing a subsequent audit. However, by deleting only a single ciphertext in one of the cuckoo tables, the attacker now deleted only a single codeword symbol, not a full block of n of them. And now we can show that our extractor can still recover enough of the other symbols of the codeword block so that the erasure code will enable recovery of the original data. Of course, the server could start deleting more of the locations in the lowest level cuckoo table, but he cannot selectively target codeword symbols belonging to a single codeword block, since it has no idea where those reside. If he starts to delete too many of them just to make sure a message block is not recoverable, then he will lose his ability to pass an audit.

Hardness of SIS and LWE with Small Parameters

Daniele Micciancio*

Chris Peikert[†]

February 13, 2013

Abstract

The Short Integer Solution (SIS) and Learning With Errors (LWE) problems are the foundations for countless applications in lattice-based cryptography, and are provably as hard as approximate lattice problems in the worst case. A important question from both a practical and theoretical perspective is how small their parameters can be made, while preserving their hardness.

We prove two main results on SIS and LWE with small parameters. For SIS, we show that the problem retains its hardness for moduli $q \geq \beta \cdot n^\delta$ for any constant $\delta > 0$, where β is the bound on the Euclidean norm of the solution. This improves upon prior results which required $q \geq \beta \cdot \sqrt{n \log n}$, and is essentially optimal since the problem is trivially easy for $q \leq \beta$. For LWE, we show that it remains hard even when the errors are small (e.g., uniformly random from $\{0, 1\}$), provided that the number of samples is small enough (e.g., linear in the dimension n of the LWE secret). Prior results required the errors to have magnitude at least \sqrt{n} and to come from a Gaussian-like distribution.

1 Introduction

In modern lattice-based cryptography, two average-case computational problems serve as the foundation of almost all cryptographic schemes: Short Integer Solution (SIS), and Learning With Errors (LWE). The SIS problem dates back to Ajtai's pioneering work [1], and is defined as follows. Let n and q be integers, where n is the primary security parameter and usually $q = \text{poly}(n)$, and let $\beta > 0$. Given a uniformly random matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ for some $m = \text{poly}(n)$, the goal is to find a nonzero integer vector $\mathbf{z} \in \mathbb{Z}^m$ such that $\mathbf{A}\mathbf{z} = \mathbf{0} \bmod q$ and $\|\mathbf{z}\| \leq \beta$ (where $\|\cdot\|$ denotes Euclidean norm). Observe that β should be set large enough to ensure that a solution exists (e.g., $\beta > \sqrt{n \log q}$ suffices), but that $\beta \geq q$ makes the problem trivially easy to solve. Ajtai showed that for appropriate parameters, SIS enjoys a remarkable worst-case/average-case hardness property: solving it *on the average* (with any noticeable probability) is at least as hard as approximating several lattice problems on n -dimensional lattices *in the worst case*, to within $\text{poly}(n)$ factors.

*University of California, San Diego. 9500 Gilman Dr., Mail Code 0404, La Jolla, CA 92093, USA. Email: daniele@cs.ucsd.edu. This material is based on research sponsored by DARPA under agreement number FA8750-11-C-0096 and NSF under grant CNS-1117936. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA, NSF or the U.S. Government.

[†]School of Computer Science, Georgia Institute of Technology. This material is based upon work supported by the National Science Foundation under CAREER Award CCF-1054495, by DARPA under agreement number FA8750-11-C-0096, and by the Alfred P. Sloan Foundation.

The LWE problem was introduced in the celebrated work of Regev [24], and has the same parameters n and q , along with a “noise rate” $\alpha \in (0, 1)$. The problem (in its search form) is to find a secret vector $\mathbf{s} \in \mathbb{Z}_q^n$, given a “noisy” random linear system $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, $\mathbf{b} = \mathbf{A}^T \mathbf{s} + \mathbf{e} \bmod q$, where \mathbf{A} is uniformly random and the entries of \mathbf{e} are i.i.d. from a Gaussian-like distribution with standard deviation roughly αq . Regev showed that as long as $\alpha q \geq 2\sqrt{n}$, solving LWE on the average (with noticeable probability) is at least as hard as approximating lattice problems in the worst case to within $\tilde{O}(n/\alpha)$ factors using a *quantum* algorithm. Subsequently, Peikert [21] gave a *classical* reduction for a subset of the lattice problems and the same approximation factors, but under the additional condition that $q \geq 2^{n/2}$ (or $q \geq 2\sqrt{n}/\alpha$ based on some non-standard lattice problems).

A significant line of research has been devoted to improving the tightness of worst-case/average-case connections for lattice problems. For SIS, a series of works [1, 7, 14, 19, 12] gave progressively better parameters that guarantee hardness, and smaller approximation factors for the underlying lattice problems. The state of the art (from [12], building upon techniques introduced in [19]) shows that for $q \geq \beta \cdot \omega(\sqrt{n \log n})$, finding a SIS solution with norm bounded by β is as hard as approximating worst-case lattice problems to within $\tilde{O}(\beta\sqrt{n})$ factors. (The parameter m does not play any significant role in the hardness results, and can be any polynomial in n .) For LWE, Regev’s initial result remains the tightest, and the requirement that $q \geq \sqrt{n}/\alpha$ (i.e., that the errors have magnitude at least \sqrt{n}) is in some sense optimal: a clever algorithm due to Arora and Ge [2] solves LWE in time $2^{\tilde{O}(\alpha q)^2}$, so a proof of hardness for substantially smaller errors would imply a subexponential time (quantum) algorithm for approximate lattice problems, which would be a major breakthrough. Interestingly, the current modulus bound for LWE is in some sense better than the one for SIS by a $\tilde{\Omega}(\sqrt{n})$ factor: there are applications of LWE for $1/\alpha = \tilde{O}(1)$ and hence $q = \tilde{O}(\sqrt{n})$, whereas SIS is only useful for $\beta \geq \sqrt{n}$, and therefore requires $q \geq n$ according to the state-of-the-art reductions.

Further investigating the smallest parameters for which SIS and LWE remain provably hard is important from both a practical and theoretical perspective. On the practical side, improvements would lead to smaller cryptographic keys without compromising the theoretical security guarantees, or may provide greater confidence in more practical parameter settings that so far lack provable hardness. Also, proving the hardness of LWE for non-Gaussian error distributions (e.g., uniform over a small set) would make applications easier to implement. Theoretically, improvements may eventually shed light on related problems like Learning Parity with Noise (LPN), which can be seen as a special case of LWE for modulus $q = 2$, and which is widely used in coding-based cryptography, but which has no known proof of hardness.

1.1 Our Results

We prove two complementary results on the hardness of SIS and LWE with small parameters. For SIS, we show that the problem retains its hardness for moduli q nearly equal to the solution bound β . For LWE, we show that it remains hard even when the errors are small (e.g., uniformly random from $\{0, 1\}$), provided that the number m of noisy equations is small enough. This qualification is necessary in light of the Arora-Ge attack [2], which for large enough m can solve LWE with binary errors in polynomial time. Details follow.

SIS with small modulus. Our first theorem says that SIS retains its hardness with a modulus as small as $q \geq \beta \cdot n^\delta$, for any $\delta > 0$. Recall that the best previous reduction [12] required $q \geq \beta \cdot \omega(\sqrt{n \log n})$, and that SIS becomes trivially easy for $q \leq \beta$, so the q obtained by our proof is essentially optimal. It also essentially closes the gap between LWE and SIS, in terms of how small a useful modulus can be. More precisely, the following is a special case of our main SIS hardness theorem; see Section 3 for full details.

Theorem 1.1 (Corollary of Theorem 3.8). *Let n and $m = \text{poly}(n)$ be integers, let $\beta \geq \beta_\infty \geq 1$ be reals, let $Z = \{\mathbf{z} \in \mathbb{Z}^m : \|\mathbf{z}\|_2 \leq \beta \text{ and } \|\mathbf{z}\|_\infty \leq \beta_\infty\}$, and let $q \geq \beta \cdot n^\delta$ for some constant $\delta > 0$. Then solving (on the average, with non-negligible probability) SIS with parameters n, m, q and solution set $Z \setminus \{\mathbf{0}\}$ is at least as hard as approximating lattice problems in the worst case on n -dimensional lattices to within $\gamma = \max\{1, \beta \cdot \beta_\infty / q\} \cdot \tilde{O}(\beta\sqrt{n})$ factors.*

Of course, the ℓ_∞ bound on the SIS solutions can be easily removed simply setting $\beta_\infty = \beta$, so that $\|\mathbf{z}\|_\infty \leq \|\mathbf{z}\|_2 \leq \beta$ automatically holds true. We include an explicit ℓ_∞ bound $\beta_\infty \leq \beta$ in order to obtain more precise hardness results, based on potentially smaller worst-case approximation factors γ . We point out that the bound β_∞ and the associated extra term $\max\{1, \beta \cdot \beta_\infty / q\}$ in the worst-case approximation factor is not present in previous results. Notice that this term can be as small as 1 (if we take $q \geq \beta \cdot \beta_\infty$, and in particular if $\beta_\infty \leq n^\delta$), and as large as β/n^δ (if $\beta_\infty = \beta$). This may be seen as the first theoretical evidence that, at least when using a small modulus q , restricting the ℓ_∞ norm of the solutions may make the SIS problem qualitatively harder than just restricting the ℓ_2 norm. There is already significant empirical evidence for this belief: the most practically efficient attacks on SIS, which use lattice basis reduction (e.g., [11, 8]), only find solutions with bounded ℓ_2 norm, whereas combinatorial attacks such as [5, 25] (see also [20]) or theoretical lattice attacks [9] that can guarantee an ℓ_∞ bound are much more costly in practice, and also require exponential space. Finally, we mention that setting $\beta_\infty \ll \beta$ is very natural in the usual formulations of one-way and collision-resistant hash functions based on SIS, where collisions correspond (for example) to vectors in $\{-1, 0, 1\}^m$, and therefore have ℓ_∞ bound $\beta_\infty = 1$, but ℓ_2 bound $\beta = \sqrt{m}$. Similar gaps between β_∞ and β can easily be enforced in other applications, e.g., digital signatures [12].

LWE with small errors. In the case of LWE, we prove a general theorem offering a trade-off among several different parameters, including the size of the errors, the dimension and number of samples in the LWE problem, and the dimension of the underlying worst-case lattice problems. Here we mention just one instantiation for the case of prime modulus and uniformly distributed *binary* (i.e., 0-1) errors, and refer the reader to Section 4 and Theorem 4.6 for the more general statement and a discussion of the parameters.

Theorem 1.2 (Corollary of Theorem 4.6). *Let n and $m = n \cdot (1 + \Omega(1/\log n))$ be integers, and $q \geq n^{O(1)}$ a sufficiently large polynomially bounded (prime) modulus. Then solving LWE with parameters n, m, q and independent uniformly random binary errors (i.e., in $\{0, 1\}$) is at least as hard as approximating lattice problems in the worst case on $\Theta(n/\log n)$ -dimensional lattices within a factor $\gamma = \tilde{O}(\sqrt{n} \cdot q)$.*

We remark that our results (see Theorem 4.6) apply to many other settings, including error vectors $\mathbf{e} \in X$ chosen from any (sufficiently large) subset $X \subseteq \{0, 1\}^m$ of binary strings, as well as error vectors with larger entries. Interestingly, our hardness result for LWE with very small errors relies on the worst-case hardness of lattice problems in dimension $n' = O(n/\log n)$, which is smaller than (but still quasi-linear in) the dimension n of the LWE problem; however, this is needed only when considering very small error vectors. Theorem 4.6 also shows that if \mathbf{e} is chosen uniformly at random with entries bounded by n^ϵ (which is still much smaller than \sqrt{n}), then the dimension of the underlying worst-case lattice problems (and the number $m - n$ of extra samples, beyond the LWE dimension n) can be linear in n .

The restriction that the number of LWE samples $m = O(n)$ be linear in the dimension of the secret can also be relaxed slightly. But some restriction is necessary, because LWE with small errors can be solved in polynomial time when given an arbitrarily large polynomial number of samples. We focus on linear $m = O(n)$ because this is enough for most (but not all) applications in lattice cryptography, including identity-based encryption and fully homomorphic encryption, when the parameters are set appropriately. (The one exception that we know of is the security proof for pseudorandom functions [3].)

1.2 Techniques and Comparison to Related Work

Our results for SIS and LWE are technically disjoint, and all they have in common is the goal of proving hardness results for smaller values of the parameters. So, we describe our technical contributions in the analysis of these two problems separately.

SIS with small modulus. For SIS, as a warm-up, we first give a proof for a special case of the problem where the input is restricted to vectors of a special form (e.g., binary vectors). For this restricted version of SIS, we are able to give a self-reduction (from SIS to SIS) which reduces the size of the modulus. So, we can rely on previous worst-case to average-case reductions for SIS as “black boxes,” resulting in an extremely simple proof. However, this simple self-reduction has some drawbacks. Beside the undesirable restriction on the SIS inputs, our the reduction is rather loose with respect to the underlying worst-case lattice approximation problem: in order to establish the hardness of SIS with small moduli q (and restricted inputs), one needs to assume the worst-case hardness of lattice problems for rather large polynomial approximation factors. (By contrast, previous hardness results for larger moduli [19, 12] only assumed hardness for quasi-linear approximation factors.) We address both drawbacks by giving a direct reduction from worst-case lattice problems to SIS with small modulus. This is our main SIS result, and it combines ideas from previous work [19, 12] with two new technical ingredients:

- All previous SIS hardness proofs [1, 7, 14, 19, 12] solved worst-case lattice problems by iteratively finding (sets of linearly independent) lattice vectors of shorter and shorter length. Our first new technical ingredient (inspired by the pioneering work of Regev [24] on LWE) is the use a different intermediate problem: instead of finding progressively shorter lattice vectors, we consider the problem of sampling lattice vectors according to Gaussian-like distributions of progressively smaller widths. To the best of our knowledge, this is the first use of Gaussian lattice sampling as an intermediate worst-case problem in the study of SIS, and it appears necessary to lower the SIS modulus below n . We mention that Gaussian lattice sampling has been used before to reduce the modulus in hardness reductions for SIS [12], but still within the framework of iteratively finding short vectors (which in [12] are used to generate fresh Gaussian samples for the reduction), which results in larger moduli $q > n$.
- The use of Gaussian lattice sampling as an intermediate problem within the SIS hardness proof yields linear combinations of several discrete Gaussian samples with adversarially chosen coefficients. Our second technical ingredient, used to analyze these linear combinations, is a new convolution theorem for discrete Gaussians (Theorem 3.3), which strengthens similar ones previously proved in [22, 6]. Here again, the strength of our new convolution theorem appears necessary to obtain hardness results for SIS with modulus smaller than n .

Our new convolution theorem may be of independent interest, and might find applications in the analysis of other lattice algorithms.

LWE with small errors. We now move to our results on LWE. For this problem, the best provably hard parameters to date were those obtained in the original paper of Regev [24], which employed Gaussian errors, and required them to have (expected) magnitude at least \sqrt{n} . These results were believed to be optimal due to a clever algorithm of Arora and Ge [2], which solves LWE in subexponential time when the errors are asymptotically smaller than \sqrt{n} . The possibility of circumventing this barrier by limiting the number of LWE samples was first suggested by Micciancio and Mol [17], who gave “sample preserving” search-to-decision reductions for LWE, and asked if LWE with small uniform errors could be proved hard when the number

of available samples is sufficiently small. Our results provide a first answer to this question, and employ concepts and techniques from the work of Peikert and Waters [23] (see also [4]) on *lossy* (trapdoor) functions. In brief, a lossy function family is an indistinguishable pair of function families \mathcal{F}, \mathcal{L} such that functions in \mathcal{F} are injective and those in \mathcal{L} are lossy, in the sense that they map their common domain to much smaller sets, and therefore lose information about the input. As shown in [23], from the indistinguishability of \mathcal{F} and \mathcal{L} , it follows that the families \mathcal{F} and \mathcal{L} are both one-way.

In Section 2 we present a generalized framework for the study of lossy function families, which does not require the functions to have trapdoors, and applies to arbitrary (not necessarily uniform) input distributions. While the techniques we use are all standard, and our definitions are minor generalizations of the ones given in [23], we believe that our framework provides a conceptual simplification of previous work, relating the relatively new notion of lossy functions to the classic security definitions of second-preimage resistance and uninvertibility.

The lossy function framework is used to prove the hardness of LWE with small uniform errors and (necessarily) a small number of samples. Specifically, we use the standard LWE problem (with large Gaussian errors) to set up a lossy function family \mathcal{F}, \mathcal{L} . (Similar families with trapdoors were constructed in [23, 4], but not for the parameterizations required to obtain interesting hardness results for LWE.) The indistinguishability of \mathcal{F} and \mathcal{L} follows directly from the hardness of the underlying LWE problem. The new hardness result for LWE (with small errors) is equivalent to the one-wayness of \mathcal{F} , and is proved by a relatively standard analysis of the second-preimage resistance and uninvertibility of certain subset-sum functions associated to \mathcal{L} .

Comparison to related work. In an independent work that was submitted concurrently with ours, Döttling and Müller-Quade [10] also used a lossiness argument to prove new hardness results for LWE. (Their work does not address the SIS problem.) At a syntactic level, they use LWE (i.e., generating matrix) notation and a new concept they call “lossy codes,” while here we use SIS (i.e., parity-check matrix) notation and rely on the standard notions of uninvertible and second-preimage resistant functions. By the dual equivalence of SIS and LWE [15, 17] (see Proposition 2.9), this can be considered a purely syntactic difference, and the high-level lossiness strategy (including the lossy function family construction) used in [10] and in our work are essentially the same. However, the low-level analysis techniques and final results are quite different. The main result proved in [10] is essentially the following.

Theorem 1.3 ([10]). *Let $n, q, m = n^{O(1)}$ and $r \geq n^{1/2+\epsilon} \cdot m$ be integers, for an arbitrary small constant $\epsilon > 0$. Then the LWE problem with parameters n, m, q and independent uniformly distributed errors in $\{-r, \dots, r\}^m$ is at least as hard as (quantumly) solving worst-case problems on $(n/2)$ -dimensional lattices to within a factor $\gamma = n^{1+\epsilon} \cdot mq/r$.*

The contribution of [10] over previous work is to prove the hardness of LWE for *uniformly distributed* errors, as opposed to errors that follow a Gaussian distribution. Notice that the magnitude of the errors used in [10] is always at least $\sqrt{n} \cdot m$, which is substantially larger (by a factor of m) than in previous results. So, [10] makes no progress towards reducing the magnitude of the errors, which is the main goal of this paper. By contrast, our work shows the hardness of LWE for errors smaller than \sqrt{n} (indeed, as small as $\{0, 1\}$), provided the number of samples is sufficiently small.

Like our work, [10] requires the number of LWE samples m to be fixed in advance (because the error magnitude r depends on m), but it allows m to be an arbitrary polynomial in n . This is possible because for the large errors $r \gg \sqrt{n}$ considered in [10], the attack of [2] runs in at least exponential time. So, in principle, it may even be possible (and is an interesting open problem) to prove the hardness of LWE with

(large) uniform errors as in [10], but for an unbounded number of samples. In our work, hardness of LWE for errors smaller than \sqrt{n} is proved for a much smaller number of samples m , and this is necessary in order to avoid the subexponential time attack of [2].

While the focus of our work is on LWE with small errors, we remark that our main LWE hardness result (Theorem 4.6) can also be instantiated using large polynomial errors $r = n^{O(1)}$ to obtain any (linear) number of samples $m = \Theta(n)$. In this setting, [10] provides a much better dependency between the magnitude of the errors and the number of samples (which in [10] can be an arbitrary polynomial). This is due to substantial differences in the low-level techniques employed in [10] and in our work to analyze the statistical properties of the lossy function family. For these same reasons, even for large errors, our results seem incomparable to those of [10] because we allow for a much wider class of error distributions.

2 Preliminaries

We use uppercase roman letters F, X for sets, lowercase roman for set elements $x \in X$, bold $\mathbf{x} \in X^n$ for vectors, and calligraphic letters $\mathcal{F}, \mathcal{X}, \dots$ for probability distributions. The support of a probability distribution \mathcal{X} is denoted $\text{supp}(\mathcal{X})$. The uniform distribution over a finite set X is denoted $\mathcal{U}(X)$.

Two probability distributions \mathcal{X} and \mathcal{Y} are (t, ϵ) -indistinguishable if for all (probabilistic) algorithms \mathcal{D} running in time at most t ,

$$|\Pr[x \leftarrow \mathcal{X} : \mathcal{D}(x) \text{ accepts}] - \Pr[y \leftarrow \mathcal{Y} : \mathcal{D}(y) \text{ accepts}]| \leq \epsilon.$$

2.1 One-Way Functions

A function family is a probability distribution \mathcal{F} over a set of functions $F \subseteq (X \rightarrow Y)$ with common domain X and range Y . Formally, function families are defined as distributions over bit strings (function descriptions) together with an evaluation algorithm, mapping each bitstring to a corresponding function, with possibly multiple descriptions associated to the same function. In this paper, for notational simplicity, we identify functions and their description, and unless stated otherwise, all statements about function families should be interpreted as referring to the corresponding probability distributions over function descriptions. For example, if we say that two function families \mathcal{F} and \mathcal{G} are indistinguishable, we mean that no efficient algorithm can distinguish between function descriptions selected according to either \mathcal{F} or \mathcal{G} , where \mathcal{F} and \mathcal{G} are probability distributions over bitstrings that are interpreted as functions using the same evaluation algorithm.

A function family \mathcal{F} is (t, ϵ) *collision resistant* if for all (probabilistic) algorithms \mathcal{A} running in time at most t ,

$$\Pr[f \leftarrow \mathcal{F}, (x, x') \leftarrow \mathcal{A}(f) : f(x) = f(x') \wedge x \neq x'] \leq \epsilon.$$

Let \mathcal{X} be a probability distribution over the domain X of a function family \mathcal{F} . We recall the following standard security notions:

- $(\mathcal{F}, \mathcal{X})$ is (t, ϵ) -*one-way* if for all probabilistic algorithms \mathcal{A} running in time at most t ,

$$\Pr[f \leftarrow \mathcal{F}, x \leftarrow \mathcal{X} : \mathcal{A}(f, f(x)) \in f^{-1}(f(x))] \leq \epsilon.$$

- $(\mathcal{F}, \mathcal{X})$ is (t, ϵ) -*uninvertible* if for all probabilistic algorithms \mathcal{A} running in time at most t ,

$$\Pr[f \leftarrow \mathcal{F}, x \leftarrow \mathcal{X} : \mathcal{A}(f, f(x)) = x] \leq \epsilon.$$

- $(\mathcal{F}, \mathcal{X})$ is (t, ϵ) -second preimage resistant if for all probabilistic algorithms \mathcal{A} running in time at most t ,

$$\Pr[f \leftarrow \mathcal{F}, x \leftarrow \mathcal{X}, x' \leftarrow \mathcal{A}(f, x) : f(x) = f(x') \wedge x \neq x'] \leq \epsilon.$$

- $(\mathcal{F}, \mathcal{X})$ is (t, ϵ) -pseudorandom if the distributions $\{f \leftarrow \mathcal{F}, x \leftarrow \mathcal{X} : (f, f(x))\}$ and $\{f \leftarrow \mathcal{F}, y \leftarrow \mathcal{U}(Y) : (f, y)\}$ are (t, ϵ) -indistinguishable.

The above probabilities (or the absolute difference between probabilities, for indistinguishability) are called the *advantages* in breaking the corresponding security notions. It easily follows from the definition that if a function family is one-way with respect to any input distribution \mathcal{X} , then it is also uninvertible with respect to the same input distribution \mathcal{X} . Also, if a function family is collision resistant, then it is also second preimage resistant with respect to any efficiently samplable input distribution.

All security definitions are immediately adapted to the asymptotic setting, where we implicitly consider sequences of finite function families indexed by a security parameter. In this setting, for any security definition (one-wayness, collision resistance, etc.) we omit t , and simply say that a function is secure if for any t that is polynomial in the security parameter, it is (t, ϵ) -secure for some ϵ that is negligible in the security parameter. We say that a function family is *statistically* secure if it is (t, ϵ) -secure for some negligible ϵ and *arbitrary* t , i.e., it is secure even with respect to computationally unbounded adversaries.

The composition of function families is defined in the natural way. Namely, for any two function families with $[\mathcal{F}] \subseteq X \rightarrow Y$ and $[\mathcal{G}] \subseteq Y \rightarrow Z$, the composition $\mathcal{G} \circ \mathcal{F}$ is the function family that selects $f \leftarrow \mathcal{F}$ and $g \leftarrow \mathcal{G}$ independently at random, and outputs the function $(g \circ f) : X \rightarrow Z$.

2.2 Lossy Function Families

Lossy functions, introduced in [23], are usually defined in the context of trapdoor function families, where the functions are efficiently invertible with the help of some trapdoor information, and therefore injective (at least with high probability over the choice of the key). We give a more general definition of lossy function families that applies to non-injective functions and arbitrary input distributions, though we will be mostly interested in input distributions that are uniform over some set.

Definition 2.1. Let \mathcal{L}, \mathcal{F} be two probability distributions (with possibly different supports) over the same set of (efficiently computable) functions $F \subseteq X \rightarrow Y$, and let \mathcal{X} be an efficiently sampleable distribution over the domain X . We say that $(\mathcal{L}, \mathcal{F}, \mathcal{X})$ is a lossy function family if the following properties are satisfied:

- the distributions \mathcal{L} and \mathcal{F} are indistinguishable,
- $(\mathcal{L}, \mathcal{X})$ is uninvertible, and
- $(\mathcal{F}, \mathcal{X})$ is second preimage resistant.

The uninvertibility and second preimage resistance properties can be either computational or statistical. (The definition from [23] requires both to be statistical.) We remark that uninvertible functions and second preimage resistant functions are not necessarily one-way. For example, the constant function $f(x) = 0$ is (statistically) uninvertible when $|X|$ is super-polynomial in the security parameter, and the identity function $f(x) = x$ is (statistically) second preimage resistant (in fact, even collision resistant), but neither is one-way. Still, if a function family is simultaneously uninvertible and second preimage resistant, then one-wayness easily follows.

Lemma 2.2. Let \mathcal{F} be a family of functions computable in time t' . If $(\mathcal{F}, \mathcal{X})$ is both (t, ϵ) -uninvertible and $(t + t', \epsilon')$ -second preimage resistant, then it is also $(t, \epsilon + \epsilon')$ -one-way.

Proof. Let \mathcal{A} be an algorithm running in time at most t and attacking the one-wayness property of $(\mathcal{F}, \mathcal{X})$. Let $f \leftarrow \mathcal{F}$ and $x \leftarrow \mathcal{X}$ be chosen at random, and compute $y \leftarrow \mathcal{A}(f, f(x))$. We want to bound the probability that $f(x) = f(y)$. We consider two cases:

- If $x = y$, then \mathcal{A} breaks the uninvertibility property of $(\mathcal{F}, \mathcal{X})$.
- If $x \neq y$, then $\mathcal{A}'(f, x) = \mathcal{A}(f, f(x))$ breaks the second preimage property of $(\mathcal{F}, \mathcal{X})$.

By assumption, the probability of these two events are at most ϵ and ϵ' respectively. By the union bound, \mathcal{A} breaks the one-wayness property with advantage at most $\epsilon + \epsilon'$. \square

It easily follows by a simple indistinguishability argument that if $(\mathcal{L}, \mathcal{F}, \mathcal{X})$ is a lossy function family, then both $(\mathcal{L}, \mathcal{X})$ and $(\mathcal{F}, \mathcal{X})$ are one-way.

Lemma 2.3. *Let \mathcal{F} and \mathcal{F}' be any two indistinguishable, efficiently computable function families, and let \mathcal{X} be an efficiently sampleable input distribution. Then if $(\mathcal{F}, \mathcal{X})$ is uninvertible (respectively, second-preimage resistant), then $(\mathcal{F}', \mathcal{X})$ is also uninvertible (resp., second-preimage resistant). In particular, if $(\mathcal{L}, \mathcal{F}, \mathcal{X})$ is a lossy function family, then $(\mathcal{L}, \mathcal{X})$ and $(\mathcal{F}, \mathcal{X})$ are both one-way.*

Proof. Assume that $(\mathcal{F}, \mathcal{X})$ is uninvertible and that there exists an efficient algorithm \mathcal{A} breaking the uninvertibility property of $(\mathcal{F}', \mathcal{X})$. Then \mathcal{F} and \mathcal{F}' can be efficiently distinguished by the following algorithm $\mathcal{D}(f)$: choose $x \leftarrow \mathcal{X}$, compute $x' \leftarrow \mathcal{A}(f, f(x))$, and accept if \mathcal{A} succeeded, i.e., if $x = x'$.

Next, assume that $(\mathcal{F}, \mathcal{X})$ is second preimage resistant, and that there exists an efficient algorithm \mathcal{A} breaking the second preimage resistance property of $(\mathcal{F}', \mathcal{X})$. Then \mathcal{F} and \mathcal{F}' can be efficiently distinguished by the following algorithm $\mathcal{D}(f)$: choose $x \leftarrow \mathcal{X}$, compute $x' \leftarrow \mathcal{A}(f, x)$, and accept if \mathcal{A} succeeded, i.e., if $x \neq x'$ and $f(x) = f(x')$.

It follows that if $(\mathcal{L}, \mathcal{F}, \mathcal{X})$ is a lossy function family, then $(\mathcal{L}, \mathcal{X})$ and $(\mathcal{F}, \mathcal{X})$ are both uninvertible and second preimage resistant. Therefore, by Lemma 2.2, they are also one-way. \square

The standard definition of (injective) lossy trapdoor functions [23], is usually stated by requiring the ratio $|f(X)|/|X|$ to be small. Our general definition can easily be related to the standard definition by specializing it to uniform input distributions. The next lemma gives an equivalent characterization of uninvertible functions when the input distribution is uniform.

Lemma 2.4. *Let \mathcal{L} be a family of functions on a common domain X , and let $\mathcal{X} = \mathcal{U}(X)$ the uniform input distribution over X . Then $(\mathcal{L}, \mathcal{X})$ is ϵ -uninvertible (even statistically, with respect to computationally unbounded adversaries) for $\epsilon = \mathbb{E}_{f \leftarrow \mathcal{L}}[|f(X)|/|X|]$.*

Proof. Fix a function f , and choose a random input $x \leftarrow \mathcal{X}$. The best (computationally unbounded) attack on the uninvertibility of $(\mathcal{L}, \mathcal{X})$, given input f and $y = f(x)$, outputs an $x' \in X$ such that $f(x') = y$ and the probability of x' under \mathcal{X} is maximized. Since \mathcal{X} is the uniform distribution over X , the conditional distribution of x given y is uniform over $f^{-1}(y)$, and the attack succeeds with probability $1/|f^{-1}(y)|$. Each y is output by f with probability $|f^{-1}(y)|/|X|$. So, the success probability of the attack is

$$\sum_{y \in f(X)} \frac{|f^{-1}(y)|}{|X|} \cdot \frac{1}{|f^{-1}(y)|} = \frac{|f(X)|}{|X|}.$$

Taking the expectation over the choice of f , we get that the attacker succeeds with probability ϵ . \square

We conclude this section with the observation that uninvertibility behaves as expected with respect to function composition.

Lemma 2.5. *If $(\mathcal{F}, \mathcal{X})$ is uninvertible and \mathcal{G} is any family of efficiently computable functions, then $(\mathcal{G} \circ \mathcal{F}, \mathcal{X})$ is also uninvertible.*

Proof. Any inverter \mathcal{A} for $\mathcal{G} \circ \mathcal{F}$ can be easily transformed into an inverter $\mathcal{A}'(f, y)$ for $(\mathcal{F}, \mathcal{X})$ that chooses $g \leftarrow \mathcal{G}$ at random, and outputs the result of running $\mathcal{A}(g \circ f, g(y))$ \square

A similar statement holds also for one-wayness, under the additional assumption that \mathcal{G} is second preimage resistant, but it is not needed here.

2.3 Lattices and Gaussians

An n -dimensional *lattice* of *rank* k is the set Λ of integer combinations of k linearly independent vectors $\mathbf{b}_1, \dots, \mathbf{b}_k \in \mathbb{R}^n$, i.e. $\Lambda = \left\{ \sum_{i=1}^k x_i \mathbf{b}_i \mid x_i \in \mathbb{Z} \text{ for } i = 1, \dots, k \right\}$. The matrix $\mathbf{B} = [\mathbf{b}_1, \dots, \mathbf{b}_k]$ is called a *basis* for the lattice Λ . The *dual* of a (not necessarily full-rank) lattice Λ is the set $\Lambda^* = \{\mathbf{x} \in \text{span}(\Lambda) : \forall \mathbf{y} \in \Lambda, \langle \mathbf{x}, \mathbf{y} \rangle \in \mathbb{Z}\}$. In what follows, unless otherwise specified we work with *full-rank* lattices, where $k = n$.

The i th *successive minimum* $\lambda_i(\Lambda)$ is the smallest radius r such that Λ contains i linearly independent vectors of (Euclidean) length at most r . A fundamental computational problem in the study of lattice cryptography is the approximate Shortest Independent Vectors Problem SIVP_γ , which, on input a full-rank n -dimensional lattice Λ (typically represented by a basis), asks to find n linearly independent lattice vectors $\mathbf{v}_1, \dots, \mathbf{v}_n \in \Lambda$ all of length at most $\gamma \cdot \lambda_n(\Lambda)$, where $\gamma \geq 1$ is an approximation factor and is usually a function of the lattice dimension n . Another problem is the (decision version of the) approximate Shortest Vector Problem GapSVP_γ , which, on input an n -dimensional lattice Λ , asks to output “yes” if $\lambda_1(\Lambda) \leq 1$ and “no” if $\lambda_1(\Lambda) > \gamma$. (If neither is the case, any answer is acceptable.)

For a matrix $\mathbf{B} = [\mathbf{b}_1, \dots, \mathbf{b}_k]$ of linearly independent vectors, the *Gram-Schmidt* orthogonalization $\tilde{\mathbf{B}}$ is the matrix of vectors $\tilde{\mathbf{b}}_i$ where $\tilde{\mathbf{b}}_1 = \mathbf{b}_1$, and for each $i = 2, \dots, k$, the vector $\tilde{\mathbf{b}}_i$ is the projection of \mathbf{b}_i orthogonal to $\text{span}(\mathbf{b}_1, \dots, \mathbf{b}_{i-1})$. The *Gram-Schmidt* minimum of a lattice Λ is $\tilde{b}l(\Lambda) = \min_{\mathbf{B}} \|\tilde{\mathbf{B}}\|$, where $\|\tilde{\mathbf{B}}\| = \max_i \|\tilde{\mathbf{b}}_i\|$ and the minimum is taken over all bases \mathbf{B} of Λ . Given any basis \mathbf{D} of a lattice Λ and any set \mathbf{S} of linearly independent vectors in Λ , it is possible to efficiently construct a basis \mathbf{B} of Λ such that $\|\tilde{\mathbf{B}}\| \leq \|\tilde{\mathbf{S}}\|$ (see [16]).

The *Gaussian function* $\rho_s: \mathbb{R}^m \rightarrow \mathbb{R}$ with parameter s is defined as $\rho_s(\mathbf{x}) = \exp(-\pi \|\mathbf{x}\|^2 / s^2)$. When s is omitted, it is assumed to be 1. The *discrete Gaussian distribution* $D_{\Lambda+\mathbf{c}, s}$ with parameter s over a lattice coset $\Lambda + \mathbf{c}$ is the distribution that samples each element $\mathbf{x} \in \Lambda + \mathbf{c}$ with probability $\rho_s(\mathbf{x}) / \rho_s(\Lambda + \mathbf{c})$, where $\rho_s(\Lambda + \mathbf{c}) = \sum_{\mathbf{y} \in \Lambda + \mathbf{c}} \rho_s(\mathbf{y})$ is a normalization factor.

For any $\epsilon > 0$, the *smoothing parameter* $\eta_\epsilon(\Lambda)$ [19] is the smallest $s > 0$ such that $\rho_{1/s}(\Lambda^* \setminus \{\mathbf{0}\}) \leq \epsilon$. When ϵ is omitted, it is some unspecified negligible function $\epsilon = n^{-\omega(1)}$ of the lattice dimension or security parameter n , which may vary from place to place.

We observe that the smoothing parameter satisfies the following decomposition lemma. The general case for the sum of several lattices (whose linear spans have trivial pairwise intersections) follows immediately by induction.

Lemma 2.6. *Let lattice $\Lambda = \Lambda_1 + \Lambda_2$ be the (internal direct) sum of two lattices such that $\text{span}(\Lambda_1) \cap \text{span}(\Lambda_2) = \{\mathbf{0}\}$, and let $\tilde{\Lambda}_2$ be the projection of Λ_2 orthogonal to $\text{span}(\Lambda_1)$. Then for any $\epsilon_1, \epsilon_2, \epsilon > 0$ such*

that $1 + \epsilon = (1 + \epsilon_1)(1 + \epsilon_2)$, we have

$$\eta_\epsilon(\tilde{\Lambda}_2) \leq \eta_\epsilon(\Lambda) \leq \eta_\epsilon(\Lambda_1 + \tilde{\Lambda}_2) \leq \max\{\eta_{\epsilon_1}(\Lambda_1), \eta_{\epsilon_2}(\tilde{\Lambda}_2)\}.$$

Proof. Let Λ^* , Λ_1^* and $\tilde{\Lambda}_2^*$ be the dual lattices of Λ , Λ_1 and $\tilde{\Lambda}_2$, respectively. For the first inequality, notice that $\tilde{\Lambda}_2^*$ is a sublattice of Λ^* . Therefore, $\rho_{1/s}(\tilde{\Lambda}_2^* \setminus \{0\}) \leq \rho_{1/s}(\Lambda^* \setminus \{0\})$ for any $s > 0$, and thus $\eta_\epsilon(\tilde{\Lambda}_2) \leq \eta_\epsilon(\Lambda)$.

Next we prove that $\eta_\epsilon(\Lambda) \leq \eta_\epsilon(\Lambda_1 + \tilde{\Lambda}_2)$. It is routine to verify that we can express the dual lattice Λ^* as the sum $\Lambda^* = \tilde{\Lambda}_1^* + \tilde{\Lambda}_2^*$, where $\tilde{\Lambda}_1$ is the projection of Λ_1 orthogonal to $\text{span}(\Lambda_2)$, and $\tilde{\Lambda}_1^*$ is its dual. Moreover, the projection of $\tilde{\Lambda}_1^*$ orthogonal to $\text{span}(\tilde{\Lambda}_2^*)$ is exactly Λ_1^* . For any $\tilde{\mathbf{x}}_1 \in \tilde{\Lambda}_1^*$, let $\mathbf{x}_1 \in \Lambda_1^*$ denote its projection orthogonal to $\text{span}(\tilde{\Lambda}_2^*)$. Then for any $s > 0$ we have

$$\begin{aligned} \rho_{1/s}(\Lambda^*) &= \sum_{\tilde{\mathbf{x}}_1 \in \tilde{\Lambda}_1^*} \sum_{\tilde{\mathbf{x}}_2 \in \tilde{\Lambda}_2^*} \rho_{1/s}(\tilde{\mathbf{x}}_1 + \tilde{\mathbf{x}}_2) \\ &= \sum_{\tilde{\mathbf{x}}_1 \in \tilde{\Lambda}_1^*} \sum_{\tilde{\mathbf{x}}_2 \in \tilde{\Lambda}_2^*} \rho_{1/s}(\mathbf{x}_1) \cdot \rho_{1/s}((\tilde{\mathbf{x}}_1 - \mathbf{x}_1) + \tilde{\mathbf{x}}_2) \\ &= \sum_{\tilde{\mathbf{x}}_1 \in \tilde{\Lambda}_1^*} \rho_{1/s}(\mathbf{x}_1) \cdot \rho_{1/s}((\tilde{\mathbf{x}}_1 - \mathbf{x}_1) + \tilde{\Lambda}_2^*) \\ &\leq \rho_{1/s}(\Lambda_1^*) \cdot \rho_{1/s}(\tilde{\Lambda}_2^*) = \rho_{1/s}(\Lambda_1^* + \tilde{\Lambda}_2^*) = \rho_{1/s}((\Lambda_1 + \tilde{\Lambda}_2)^*), \end{aligned}$$

where the inequality follows from the bound $\rho_{1/s}(\Lambda + \mathbf{c}) \leq \rho_{1/s}(\Lambda)$ from [19, Lemma 2.9], and the last two equalities follow from the orthogonality of Λ_1^* and $\tilde{\Lambda}_2^*$. This proves that $\eta_\epsilon(\Lambda) \leq \eta_\epsilon(\Lambda_1 + \tilde{\Lambda}_2)$.

Finally, for $s_1 = \eta_{\epsilon_1}(\Lambda_1)$, $s_2 = \eta_{\epsilon_2}(\tilde{\Lambda}_2)$ and $s = \max\{s_1, s_2\}$, we have

$$\rho_{1/s}((\Lambda_1 + \tilde{\Lambda}_2)^*) = \rho_{1/s}(\Lambda_1^*) \cdot \rho_{1/s}(\tilde{\Lambda}_2^*) \leq \rho_{1/s_1}(\Lambda_1^*) \cdot \rho_{1/s_2}(\tilde{\Lambda}_2^*) = (1 + \epsilon_1)(1 + \epsilon_2) = 1 + \epsilon.$$

Therefore, $\eta_\epsilon(\Lambda_1 + \tilde{\Lambda}_2) \leq s$. \square

Using the decomposition lemma, one easily obtains known bounds on the smoothing parameter. For example, for any lattice basis $\mathbf{B} = [\mathbf{b}_1, \dots, \mathbf{b}_n]$, applying Lemma 2.6 repeatedly to the decomposition into the rank-1 lattices defined by each of the basis vectors yields $\eta(\mathbf{B} \cdot \mathbb{Z}^n) \leq \max_i \eta(\tilde{\mathbf{b}}_i \cdot \mathbb{Z}) = \|\tilde{\mathbf{B}}\| \cdot \omega_n$, where $\omega_n = \eta(\mathbb{Z}) = \omega(\sqrt{\log n})$ is the smoothing parameter of the integer lattice \mathbb{Z} . Choosing a basis \mathbf{B} achieving $\tilde{bl}(\Lambda) = \min_{\mathbf{B}} \|\tilde{\mathbf{B}}\|$ (where the minimum is taken over all bases \mathbf{B} of Λ), we get the bound $\eta(\Lambda) \leq \tilde{bl}(\Lambda) \cdot \omega_n$ from [12, Theorem 3.1]. Similarly, choosing a set $\mathbf{S} \subset \Lambda$ of linearly independent vectors of length $\|\mathbf{S}\| \leq \lambda_n(\Lambda)$, we get the bound $\eta(\Lambda) \leq \eta(\mathbf{S} \cdot \mathbb{Z}^n) \leq \|\tilde{\mathbf{S}}\| \cdot \omega_n \leq \|\mathbf{S}\| \cdot \omega_n = \lambda_n(\Lambda) \cdot \omega_n$ from [19, Lemma 3.3]. In this paper we use a further generalization of these bounds, still easily obtained from the decomposition lemma.

Corollary 2.7. *The smoothing parameter of the tensor product of any two lattices Λ_1, Λ_2 satisfies $\eta(\Lambda_1 \otimes \Lambda_2) \leq \tilde{bl}(\Lambda_1) \cdot \eta(\Lambda_2)$.*

Proof. Let $\mathbf{B} = [\mathbf{b}_1, \dots, \mathbf{b}_k]$ be a basis of Λ_1 achieving $\max_i \|\tilde{\mathbf{b}}_i\| = \tilde{bl}(\Lambda_1)$, and consider the natural decomposition of $\Lambda_1 \otimes \Lambda_2$ into the sum

$$(\mathbf{b}_1 \otimes \Lambda_2) + \dots + (\mathbf{b}_k \otimes \Lambda_2).$$

Notice that the projection of each sublattice $\mathbf{b}_i \otimes \Lambda_2$ orthogonal to the previous sublattices $\mathbf{b}_j \otimes \Lambda_2$ (for $j < i$) is precisely $\tilde{\mathbf{b}}_i \otimes \Lambda_2$, and has smoothing parameter $\eta(\tilde{\mathbf{b}}_i \otimes \Lambda_2) = \|\tilde{\mathbf{b}}_i\| \cdot \eta(\Lambda_2)$. Therefore, by repeated application of Lemma 2.6, we have $\eta(\Lambda_1 \otimes \Lambda_2) \leq \max_i \|\tilde{\mathbf{b}}_i\| \cdot \eta(\Lambda_2) = \tilde{bl}(\Lambda_1) \cdot \eta(\Lambda_2)$. \square

The following proposition relates the problem of sampling lattice vectors according to a Gaussian distribution to the SIVP.

Proposition 2.8 ([24], Lemma 3.17). *There is a polynomial time algorithm that, given a basis for an n -dimensional lattice Λ and polynomially many samples from $D_{\Lambda, \sigma}$ for some $\sigma \geq 2\eta(\Lambda)$, solves SIVP_γ on input lattice Λ (in the worst case over Λ , and with overwhelming probability over the choice of the lattice samples) for approximation factor $\gamma = \sigma\sqrt{n} \cdot \omega_n$.*

2.4 The SIS and LWE Functions

In this paper we are interested in two special families of functions, which are the fundamental building blocks of lattice cryptography. Both families are parametrized by three integers m, n and q , and a set $X \subseteq \mathbb{Z}^m$ of short vectors. Usually n serves as a security parameter and m and q are functions of n .

The *Short Integer Solution* function family $\text{SIS}(m, n, q, X)$ is the set of all functions $f_{\mathbf{A}}$ indexed by $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ with domain $X \subseteq \mathbb{Z}^m$ and range $Y = \mathbb{Z}_q^n$, defined as $f_{\mathbf{A}}(\mathbf{x}) = \mathbf{A}\mathbf{x} \bmod q$. The *Learning With Errors* function family $\text{LWE}(m, n, q, X)$ is the set of all functions $g_{\mathbf{A}}$ indexed by $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ with domain $\mathbb{Z}_q^n \times X$ and range $Y = \mathbb{Z}_q^m$, defined as $g_{\mathbf{A}}(\mathbf{s}, \mathbf{x}) = \mathbf{A}^T \mathbf{s} + \mathbf{x} \bmod q$. Both function families are endowed with the uniform distribution over $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$. We omit the set X from the notation $\text{SIS}(m, n, q)$ and $\text{LWE}(m, n, q)$ when clear from the context, or unimportant.

In the context of collision resistance, we sometimes write $\text{SIS}(m, n, q, \beta)$ for some real $\beta > 0$, without an explicit domain X . Here the collision-finding problem is, given $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, to find distinct $\mathbf{x}, \mathbf{x}' \in \mathbb{Z}^m$ such that $\|\mathbf{x} - \mathbf{x}'\| \leq \beta$ and $f_{\mathbf{A}}(\mathbf{x}) = f_{\mathbf{A}}(\mathbf{x}')$. It is easy to see that this is equivalent to finding a nonzero $\mathbf{z} \in \mathbb{Z}^m$ of length at most $\|\mathbf{z}\| \leq \beta$ such that $f_{\mathbf{A}}(\mathbf{z}) = \mathbf{0}$.

For other security properties (e.g., one-wayness, uninvertibility, etc.), the most commonly used classes of domains and input distributions \mathcal{X} for SIS are the uniform distribution $\mathcal{U}(X)$ over the set $X = \{0, \dots, s-1\}^m$ or $X = \{-s, \dots, 0, \dots, s\}^m$, and the discrete Gaussian distribution $D_{\mathbb{Z}, s}^m$. Usually, this distribution is restricted to the set of short vectors $X = \{\mathbf{x} \in \mathbb{Z}^m : \|\mathbf{x}\| \leq s\sqrt{m}\}$, which carries all but a $2^{-\Omega(m)}$ fraction of the probability mass of $D_{\mathbb{Z}, s}^m$.

For the LWE function family, the input is usually chosen according to distribution $\mathcal{U}(\mathbb{Z}_q^n) \times \mathcal{X}$, where \mathcal{X} is one of the SIS input distributions. This makes the SIS and LWE function families essentially equivalent, as shown in the following proposition.

Proposition 2.9 ([15, 17]). *For any $n, m \geq n + \omega(\log n)$, q , and distribution \mathcal{X} over \mathbb{Z}^m , the $\text{LWE}(m, n, q)$ function family is one-way (resp. pseudorandom, or uninvertible) with respect to input distribution $\mathcal{U}(\mathbb{Z}_q^n) \times \mathcal{X}$ if and only if the $\text{SIS}(m, m - n, q)$ function family is one-way (resp. pseudorandom, or uninvertible) with respect to the input distribution \mathcal{X} .*

In applications, the SIS function family is typically used with larger input domains X for which the functions are surjective but not injective, while the LWE function family is used with smaller domains X for which the functions are injective, but not surjective. The results in this paper are more naturally stated using the SIS function family, so we will use the SIS formulation to establish our main results, and then reformulate them in terms of the LWE function family by invoking Proposition 2.9. We also use Proposition 2.9 to reformulate known hardness results (from worst-case complexity assumptions) for LWE in terms of SIS.

Assuming the quantum worst-case hardness of standard lattice problems, Regev [24] showed that the $\text{LWE}(m, n, q)$ function family is hard to invert with respect to the discrete Gaussian error distribution $D_{\mathbb{Z}, \sigma}^m$ for any $\sigma > 2\sqrt{n}$. (See also [21] for a classical reduction that requires q to be exponentially large in n .)

Because we are concerned with small parameters in this work, we focus mainly on the implications of the quantum reduction.)

Proposition 2.10 ([24], Theorem 3.1). *For any $m = n^{O(1)}$, integer q and real $\alpha \in (0, 1)$ such that $\alpha q > 2\sqrt{n}$, there is a polynomial time quantum reduction from sampling $D_{\Lambda, \sigma}$ (for any n -dimensional lattice Λ and $\sigma > (\sqrt{2n}/\alpha)\eta(\Lambda)$) to inverting the $\text{LWE}(m, n, q)$ function family on input $\mathcal{Y} = D_{\mathbb{Z}^m, \alpha q}$.*

Combining Propositions 2.8, 2.9 and 2.10, we get the following corollary.

Corollary 2.11. *For any positive m, n such that $\omega(\log n) \leq m - n \leq n^{O(1)}$ and $2\sqrt{n} < \sigma < q$, the $\text{SIS}(m, m - n, q)$ function family is uninvertible with respect to input distribution $D_{\mathbb{Z}, \sigma}^m$, under the assumption that no (quantum) algorithm can efficiently sample from a distribution statistically close to $D_{\Lambda, \sqrt{2n}q/\sigma}$.*

In particular, assuming the worst-case (quantum) hardness of $\text{SIVP}_{n\omega_n q/\sigma}$ over n -dimensional lattices, the $\text{SIS}(m, m - n, q)$ function family is uninvertible with respect to input distribution $D_{\mathbb{Z}, \sigma}^m$.

We use the fact that LWE/SIS is not only hard to invert, but also pseudorandom. This is proved using search-to-decision reductions for those problems. The most general such reductions known to date are given in the following two theorems.

Theorem 2.12 ([17]). *For any positive m, n such that $\omega(\log n) \leq m - n \leq n^{O(1)}$, any positive $\sigma \leq n^{O(1)}$, and any q with no divisors in the interval $((\sigma/\omega_n)^{m/k}, \sigma \cdot \omega_n)$, if $\text{SIS}(m, m - n, q, D_{\mathbb{Z}, \sigma}^m)$ is uninvertible, then it is also pseudorandom.*

Notice that when $\sigma > \omega_n^{(m+k)/(m-k)}$, the interval $((\sigma/\omega_n)^{m/k}, \sigma \cdot \omega_n)$ is empty, and Theorem 2.12 holds without any restriction on the factorization of the modulus q .

Theorem 2.13 ([18]). *Let q have prime factorization $q = p_1^{e_1} \cdots p_k^{e_k}$ for pairwise distinct $\text{poly}(n)$ -bounded primes p_i with each $e_i \geq 1$, and let $0 < \alpha \leq 1/\omega_n$. If $\text{LWE}(m, n, q, D_{\mathbb{Z}, \alpha q}^m)$ is hard to invert for all $m(n) = n^{O(1)}$, then $\text{LWE}(m', n, q, D_{\mathbb{Z}, \alpha' q}^m)$ is pseudorandom for any $m' = n^{O(1)}$ and*

$$\alpha' \geq \max\{\alpha, \omega_n^{1+1/\ell} \cdot \alpha^{1/\ell}, \omega_n/p_1^{e_1}, \dots, \omega_n/p_k^{e_k}\},$$

where ℓ is an upper bound on number of prime factors $p_i < \omega_n/\alpha'$.

In this work we focus on the use of Theorem 2.12, because it guarantees pseudorandomness for the *same* value of m as for the assumed one-wayness. This feature is important for applying our results from Section 4, which guarantee one-wayness for *particular* values of m (but not necessarily all $m = n^{O(1)}$).

Corollary 2.14. *For any positive m, n, σ, q such that $\omega(\log n) \leq m - n \leq n^{O(1)}$ and $2\sqrt{n} < \sigma < q < n^{O(1)}$, if q has no divisors in the range $((\sigma/\omega_n)^{1+n/k}, \sigma \cdot \omega_n)$, then the $\text{SIS}(m, m - n, q)$ function family is pseudorandom with respect to input distribution $D_{\mathbb{Z}, \sigma}^m$, under the assumption that no (quantum) algorithm can efficiently sample (up to negligible statistical errors) $D_{\Lambda, \sqrt{2n}q/\sigma}$.*

In particular, assuming the worst-case (quantum) hardness of $\text{SIVP}_{n\omega_n q/\sigma}$ on n -dimensional lattices, the $\text{SIS}(m, m - n, q)$ function family is pseudorandom with respect to input distribution $D_{\mathbb{Z}, \sigma}^m$.

3 Hardness of SIS with Small Modulus

We first prove a simple “success amplification” lemma for collision-finding in SIS, which says that any inverse-polynomial advantage can be amplified to essentially 1, at only the expense of a larger runtime and value of m (which will have no ill effects on our final results). Therefore, for the remainder of this section we implicitly restrict our attention to collision-finding algorithms that have overwhelming advantage.

Lemma 3.1. *For arbitrary n, q, m and $X \subseteq \mathbb{Z}^m$, suppose there exists a probabilistic algorithm \mathcal{A} that has advantage $\epsilon > 0$ in collision-finding for $\text{SIS}(m, n, q, X)$. Then there exists a probabilistic algorithm \mathcal{B} that has advantage $1 - (1 - \epsilon)^t \geq 1 - \exp(-\epsilon t) = 1 - \exp(-n)$ in collision-finding for $\text{SIS}(M = t \cdot m, n, q, X')$, where $t = n/\epsilon$ and $X' = \bigcup_{i=1}^t (\{0^m\}^{i-1} \times X \times \{0^m\}^{t-i})$. The runtime of \mathcal{B} is essentially t times that of \mathcal{A} .*

Proof. The algorithm \mathcal{B} simply partitions its input $\mathbf{A} \in \mathbb{Z}_q^{n \times M}$ into blocks $\mathbf{A}_i \in \mathbb{Z}_q^{n \times m}$ and invokes \mathcal{A} (with fresh random coins) on each of them, until \mathcal{A} returns a valid collision $\mathbf{x}, \mathbf{x}' \in X$ for some \mathbf{A}_i . Then \mathcal{B} returns

$$(0^{m(i-1)}, \mathbf{x}, 0^{m(t-i)}), (0^{m(i-1)}, \mathbf{x}', 0^{m(t-i)}) \in X'$$

as a collision for \mathbf{A} . Clearly, \mathcal{B} succeeds if any call to \mathcal{A} succeeds. Since all t calls to \mathcal{A} are on independent inputs \mathbf{A}_i and use independent coins, some call will succeed, except with $(1 - \epsilon)^t$ probability. \square

3.1 SIS-to-SIS Reduction

Our first proof that the $\text{SIS}(m, n, q, \beta)$ function family is collision resistant for moduli q as small as $n^{1/2+\delta}$ proceeds by a reduction between SIS problems with different parameters. Previous hardness results based on worst-case lattice assumptions require the modulus q to be at least $\beta \cdot \omega(\sqrt{n \log n})$ [12, Theorem 9.2], and $\beta \geq \sqrt{n \log q}$ is needed to guarantee that a nontrivial solution exists. For such parameters, SIS is collision resistant assuming the hardness of approximating worst-case lattice problems to within $\approx \beta \sqrt{n}$ factors.

The intuition behind our proof for smaller moduli is easily explained. We reduce SIS with modulus q^c and solution bound β^c (for any constant integer $c \geq 1$) to SIS with modulus q and bound β . Then as long as $(q/\beta)^c \geq \omega(\sqrt{n \log n})$, the former problem enjoys worst-case hardness, hence so does the latter. Thus we can take $q = \beta \cdot n^\delta$ for any constant $\delta > 0$, and $c > 1/(2\delta)$. Notice, however, that the underlying approximation factor for worst-case lattice problems is $\approx \beta^c \sqrt{n} \geq n^{1/2+1/(4\delta)}$, which, while still polynomial, degrades severely as δ approaches 0. In the next subsection we give a direct reduction from worst-case lattice problems to SIS with a small modulus, which does not have this drawback.

The above discussion is formalized in the following proposition. For technical reasons, we prove that $\text{SIS}(m, n, q, X)$ is collision resistant assuming that the domain X has the property that all SIS solutions $\mathbf{z} \in (X - X) \setminus \{\mathbf{0}\}$ satisfy $\gcd(\mathbf{z}, q) = 1$. This restriction is satisfied in many (but not all) common settings, e.g., when $q > \beta$ is prime, or when $X \subseteq \{0, 1\}^m$ is a set of binary vectors.

Proposition 3.2. *Let n, q, m, β and $X \subseteq \mathbb{Z}^m$ be such that $\gcd(\mathbf{x} - \mathbf{x}', q) = 1$ and $\|\mathbf{x} - \mathbf{x}'\| \leq \beta$ for any distinct $\mathbf{x}, \mathbf{x}' \in X$. For any positive integer c , there is a deterministic reduction from collision-finding for $\text{SIS}(m^c, n, q^c, \beta^c)$ to collision-finding for $\text{SIS}(m, n, q, X)$ (in both cases, with overwhelming advantage). The reduction runs in time polynomial in its input size, and makes fewer than m^c calls to its oracle.*

Proof. Let \mathcal{A} be an efficient algorithm that finds a collision for $\text{SIS}(m, n, q, X)$ with overwhelming advantage. We use it to find a nonzero solution for $\text{SIS}(m^c, n, q^c, \beta^c)$. Let $\mathbf{A} \in \mathbb{Z}_{q^c}^{n \times m^c}$ be an input SIS instance. Partition the columns of \mathbf{A} into m^{c-1} blocks $\mathbf{A}_i \in \mathbb{Z}_{q^c}^{n \times m}$, and for each one, invoke \mathcal{A} to find a collision modulo q , i.e., a pair of distinct vectors $\mathbf{x}_i, \mathbf{x}'_i \in X$ such that $\mathbf{A}_i \mathbf{z}_i = \mathbf{0} \pmod q$, where $\mathbf{z}_i = \mathbf{x}_i - \mathbf{x}'_i$ and $\|\mathbf{z}_i\| \leq \beta$.

For each i , since $\gcd(z_i, q) = 1$ and $\mathbf{A}_i \mathbf{z}_i = \mathbf{0} \bmod q$, the vector $\mathbf{a}'_i = (\mathbf{A}_i \mathbf{z}_i)/q \in \mathbb{Z}_{q^{c-1}}^n$ is uniformly random, even after conditioning on \mathbf{z}_i and $\mathbf{A}_i \bmod q$. So, the matrix $\mathbf{A}' \in \mathbb{Z}_{q^{c-1}}^{n \times m^{c-1}}$ made up of all these columns is uniformly random. By induction on c , using \mathcal{A} we can find a nonzero solution $\mathbf{z}' \in \mathbb{Z}^{m^{c-1}}$ such that $\mathbf{A}' \mathbf{z}' = \mathbf{0} \bmod q^{c-1}$ and $\|\mathbf{z}'\| \leq \beta^{c-1}$. Then it is easy to verify that a nonzero solution for the original instance \mathbf{A} is given by $\mathbf{z} = (z'_1 \cdot \mathbf{z}_1, \dots, z'_{m^{c-1}} \cdot \mathbf{z}_{m^{c-1}}) \in \mathbb{Z}^{m^c}$, and that $\|\mathbf{z}\| \leq \|\mathbf{z}'\| \cdot \max_i \|\mathbf{z}_i\| \leq \beta^c$. Finally, the total number of calls to \mathcal{A} is $\sum_{i=0}^{c-1} m^i < m^c$, as claimed. \square

3.2 Direct Reduction

As mentioned above, the large worst-case approximation factor associated with the use of Proposition 3.2 is undesirable, as is (to a lesser extent) the restriction that $\gcd(X - X, q) = 1$. To eliminate these drawbacks, we next give a direct proof that SIS is collision resistant for small q , based on the assumed hardness of worst-case lattice problems. The underlying approximation factor for these problems can be as small as $\tilde{O}(\beta\sqrt{n})$, which matches the best known factors obtained by previous proofs (which require a larger modulus q). Our new proof combines ideas from [19, 12] and Proposition 3.2, as well as a new convolution theorem for discrete Gaussians which strengthens similar ones previously proved in [22, 6].

Our proof of the convolution theorem is substantially different and, we believe, technically simpler than the prior ones. In particular, it handles the sum of many Gaussian samples all at once, whereas previous proofs used induction from a base case of two samples. With the inductive approach, it is technically complex to verify that all the intermediate Gaussian parameters (which involve harmonic means) satisfy the hypotheses. Moreover, the intermediate parameters can depend on the order in which the samples are added in the induction, leading to unnecessarily strong hypotheses on the original parameters.

Theorem 3.3. *Let Λ be an n -dimensional lattice, $\mathbf{z} \in \mathbb{Z}^m$ a nonzero integer vector, $s_i \geq \sqrt{2}\|\mathbf{z}\|_\infty \cdot \eta(\Lambda)$, and $\Lambda + \mathbf{c}_i$ arbitrary cosets of Λ for $i = 1, \dots, m$. Let \mathbf{y}_i be independent vectors with distributions $D_{\Lambda + \mathbf{c}_i, s_i}$, respectively. Then the distribution of $\mathbf{y} = \sum_i z_i \mathbf{y}_i$ is statistically close to $D_{Y, s}$, where $Y = \gcd(\mathbf{z})\Lambda + \mathbf{c}$, $\mathbf{c} = \sum_i z_i \mathbf{c}_i$, and $s = \sqrt{\sum_i (z_i s_i)^2}$.*

In particular, if $\gcd(\mathbf{z}) = 1$ and $\sum_i z_i \mathbf{c}_i \in \Lambda$, then \mathbf{y} is distributed statistically close to $D_{\Lambda, s}$.

Proof. First we verify that the support of \mathbf{y} is

$$\sum_i z_i (\Lambda + \mathbf{c}_i) = \sum_i z_i \Lambda + \sum_i z_i \cdot \mathbf{c}_i = \gcd(\mathbf{z})\Lambda + \sum_i z_i \cdot \mathbf{c}_i = Y.$$

So it remains to prove that each $\mathbf{y} \in Y$ has probability (nearly) proportional to $\rho_s(\mathbf{y})$.

For the remainder of the proof we use the following convenient scaling. Define the diagonal matrices $\mathbf{S} = \text{diag}(s_1, \dots, s_m)$ and $\mathbf{S}' = \mathbf{S} \otimes \mathbf{I}_n$, and the mn -dimensional lattice $\Lambda' = \bigoplus_i (s_i^{-1} \Lambda) = (\mathbf{S}')^{-1} \cdot \Lambda^{\oplus m}$, where \bigoplus denotes the (external) direct sum of lattices and $\Lambda^{\oplus m} = \mathbb{Z}^m \otimes \Lambda$ is the direct sum of m copies of Λ . Then by independence of the \mathbf{y}_i , it can be seen that $\mathbf{y}' = (\mathbf{S}')^{-1} \cdot (\mathbf{y}_1, \dots, \mathbf{y}_m)$ has discrete Gaussian distribution $D_{\Lambda' + \mathbf{c}'}$ (with parameter 1), where $\mathbf{c}' = (\mathbf{S}')^{-1} \cdot (\mathbf{c}_1, \dots, \mathbf{c}_m)$.

The output vector $\mathbf{y} = \sum_i z_i \mathbf{y}_i$ can be expressed, using the mixed-product property for Kronecker products, as

$$\mathbf{y} = (\mathbf{z}^T \otimes \mathbf{I}_n) \cdot (\mathbf{y}_1, \dots, \mathbf{y}_m) = (\mathbf{z}^T \otimes \mathbf{I}_n) \cdot \mathbf{S}' \cdot \mathbf{y}' = ((\mathbf{z}^T \mathbf{S}) \otimes \mathbf{I}_n) \cdot \mathbf{y}'.$$

So, letting $\mathbf{Z} = ((\mathbf{z}^T \mathbf{S}) \otimes \mathbf{I}_n)$, we want to prove that the distribution of $\mathbf{y} \sim \mathbf{Z} \cdot D_{\Lambda' + \mathbf{c}'}$ is statistically close to $D_{Y, s}$.

Fix any vectors $\mathbf{x}' \in \Lambda' + \mathbf{c}'$ and $\bar{\mathbf{y}} = \mathbf{Z}\mathbf{x}' \in Y$, and define the proper sublattice

$$L = \{\mathbf{v} \in \Lambda' : \mathbf{Z}\mathbf{v} = \mathbf{0}\} = \Lambda' \cap \ker(\mathbf{Z}) \subsetneq \Lambda'.$$

It is immediate to verify that the set of all $\mathbf{y}' \in \Lambda' + \mathbf{c}'$ such that $\mathbf{Z}\mathbf{y}' = \bar{\mathbf{y}}$ is $(\Lambda' + \mathbf{c}') \cap \ker(\mathbf{Z}) = L + \mathbf{x}'$. Let \mathbf{x} be orthogonal projection of \mathbf{x}' onto $\ker(\mathbf{Z}) \supset L$. Then we have

$$\Pr[\mathbf{y} = \bar{\mathbf{y}}] = \frac{\rho(L + \mathbf{x}')}{\rho(\Lambda' + \mathbf{c}')} = \rho(\mathbf{x}' - \mathbf{x}) \cdot \frac{\rho(L + \mathbf{x})}{\rho(\Lambda' + \mathbf{c}')}.$$

Below we show that $\eta(L) \leq 1$, which implies that $\rho(L + \mathbf{x})$ is essentially the same for all values of \mathbf{x}' , and hence for all $\bar{\mathbf{y}}$. Therefore, we just need to analyze $\rho(\mathbf{x}' - \mathbf{x})$.

Since \mathbf{Z}^T is an orthogonal basis for $\ker(\mathbf{Z})^\perp$, each of whose columns has Euclidean norm $s = (\sum_i (z_i s_i)^2)^{1/2}$, we have $\mathbf{x}' - \mathbf{x} = (\mathbf{Z}^T \mathbf{Z} \mathbf{x}')/s^2$, and

$$\|\mathbf{x}' - \mathbf{x}\|^2 = \langle \mathbf{x}', \mathbf{Z}^T \mathbf{Z} \mathbf{x}' \rangle / s^2 = \|\mathbf{Z} \mathbf{x}'\|^2 / s^2 = (\|\bar{\mathbf{y}}\|/s)^2.$$

Therefore, $\rho(\mathbf{x}' - \mathbf{x}) = \rho_s(\bar{\mathbf{y}})$, and so $\Pr[\mathbf{y} = \bar{\mathbf{y}}]$ is essentially proportional to $\rho_s(\bar{\mathbf{y}})$, i.e., the statistical distance between \mathbf{y} and $D_{Y,s}$ is negligible.

It remains to bound the smoothing parameter of L . Consider the m -dimensional integer lattice $Z = \mathbb{Z}^m \cap \ker(\mathbf{Z}^T) = \{\mathbf{v} \in \mathbb{Z}^m : \langle \mathbf{z}, \mathbf{v} \rangle = 0\}$. Because $(Z \otimes \Lambda) \subseteq (\mathbb{Z}^m \otimes \Lambda)$ and $\mathbf{S}^{-1}Z \subset \ker(\mathbf{z}^T \mathbf{S})$, it is straightforward to verify from the definitions that

$$(\mathbf{S}')^{-1} \cdot (Z \otimes \Lambda) = ((\mathbf{S}^{-1}Z) \otimes \Lambda)$$

is a sublattice of L . It follows from Corollary 2.7 and by scaling that

$$\eta(L) \leq \eta((\mathbf{S}')^{-1} \cdot (Z \otimes \Lambda)) \leq \eta(\Lambda) \cdot \tilde{bl}(Z) / \min s_i.$$

Finally, $\tilde{bl}(Z) \leq \min\{\|\mathbf{z}\|, \sqrt{2}\|\mathbf{z}\|_\infty\}$ because Z has a full-rank set of vectors $z_i \cdot \mathbf{e}_j - z_j \cdot \mathbf{e}_i$, where index i minimizes $|z_i| \neq 0$, and j ranges over $\{1, \dots, m\} \setminus \{i\}$. By assumption on the s_i , we have $\eta(L) \leq 1$ as desired, and the proof is complete. \square

Remark 3.4. Although we will not need it in this work, we note that the statement and proof of Theorem 3.3 can be adapted to the case where the \mathbf{y}_i respectively have *non-spherical* discrete Gaussian distributions $D_{\Lambda_i + \mathbf{c}_i, \sqrt{\Sigma_i}}$ with positive definite “covariance” parameters $\Sigma_i \in \mathbb{R}^{n \times n}$, over cosets of possibly different lattices Λ_i . (See [22] for a formal definition of these distributions.)

In this setting, by scaling Λ_i and Σ_i we can assume without loss of generality that $\mathbf{z} = (1, 1, \dots, 1)$. The theorem statement says that \mathbf{y} ’s distribution is close to a discrete Gaussian (over an appropriate lattice coset) with covariance parameter $\Sigma = \sum \Sigma_i$, under mild assumptions on $\sqrt{\Sigma_i}$. In the proof we simply let \mathbf{S}' be the block-diagonal matrix with the $\sqrt{\Sigma_i}$ as its diagonal blocks, let $\Lambda' = (\mathbf{S}')^{-1} \cdot \bigoplus_i \Lambda_i$, and let $\mathbf{Z} = (\mathbf{z}^T \otimes \mathbf{I}_n) \cdot \mathbf{S}' = [\sqrt{\Sigma_1} \mid \dots \mid \sqrt{\Sigma_m}]$. Then the only technical difference is in bounding the smoothing parameter of L .

The convolution theorem implies the following simple but useful lemma, which shows how to convert samples having a broad range of parameters into ones having parameters in a desired narrow range.

Lemma 3.5. *There is an efficient algorithm which, given a basis \mathbf{B} of some lattice Λ , some $R \geq \sqrt{2}$ and samples (\mathbf{y}_i, s_i) where each $s_i \in [\sqrt{2}, R] \cdot \eta(\Lambda)$ and each \mathbf{y}_i has distribution D_{Λ, s_i} , with overwhelming probability outputs a sample (\mathbf{y}, s) where $s \in [R, \sqrt{2}R] \cdot \eta(\Lambda)$ and \mathbf{y} has distribution statistically close to $D_{\Lambda, s}$.*

Proof. Let $\omega_n = \omega(\sqrt{\log n})$ satisfy $\omega_n \leq \sqrt{n}$. The algorithm draws $2n^2$ input samples, and works as follows: if at least n^2 of the samples have parameters $s_i \leq R \cdot \eta(\Lambda)/(\sqrt{n} \cdot \omega_n)$, then with overwhelming probability they all have lengths bounded by $R \cdot \eta(\Lambda)/\omega_n$ and they include n linearly independent vectors. Using such vectors we can construct a basis \mathbf{S} such that $\|\mathbf{S}\| \leq R \cdot \eta(\Lambda)/\omega_n$, and with the sampling algorithm of [12, Theorem 4.1] we can generate samples having parameter $R \cdot \eta(\Lambda)$.

Otherwise, at least n^2 of the samples (\mathbf{y}_i, s_i) have parameters $s_i \geq \max\{R/n, \sqrt{2}\} \cdot \eta(\Lambda)$. Then by summing an appropriate subset of those \mathbf{y}_i , by the convolution theorem we can obtain a sample having parameter in the desired range. \square

The next lemma is the heart of our reduction. The novel part, corresponding to the properties described in the second item, is a way of using a collision-finding oracle to reduce the Gaussian width of samples drawn from a lattice. The first item corresponds to the guarantees provided by previous reductions.

Lemma 3.6. *Let m, n be integers, $S = \{\mathbf{z} \in \mathbb{Z}^m \setminus \{\mathbf{0}\} \mid \|\mathbf{z}\| \leq \beta \wedge \|\mathbf{z}\|_\infty \leq \beta_\infty\}$ for some real $\beta \geq \beta_\infty > 0$, and q an integer modulus with at most $\text{poly}(n)$ integer divisors less than β_∞ . There is a probabilistic polynomial time reduction that, on input any basis \mathbf{B} of a lattice Λ and sufficiently many samples (\mathbf{y}_i, s_i) where $s_i \geq \sqrt{2}q \cdot \eta(\Lambda)$ and \mathbf{y}_i has distribution D_{Λ, s_i} , and given access to an $\text{SIS}(m, n, q, S)$ oracle (that finds collisions $\mathbf{z} \in S$ with nonnegligible probability) outputs (with overwhelming probability) a sample (\mathbf{y}, s) with $\min s_i/q \leq s \leq (\beta/q) \cdot \max s_i$, and $\mathbf{y} \in \Lambda$ such that:*

- $\mathbb{E}[\|\mathbf{y}\|] \leq (\beta\sqrt{n}/q) \cdot \max s_i$, and for any subspace $H \subset \mathbb{R}^n$ of dimension at most $n - 1$, with probability at least $1/10$ we have $\mathbf{y} \notin H$.
- Moreover, if each $s_i \geq \sqrt{2}\beta_\infty q \cdot \eta(\Lambda)$, then the distribution of \mathbf{y} is statistically close to $D_{\Lambda, s}$.

Proof. Let \mathcal{A} be the collision-finding oracle. Without loss of generality, we can assume that whenever \mathcal{A} outputs a valid collision $\mathbf{z} \in S$, we have that $\gcd(\mathbf{z})$ divides q . This is so because for any integer vector \mathbf{z} , if $\mathbf{A}\mathbf{z} = \mathbf{0} \pmod{q}$ then also $\mathbf{A}((g/d)\mathbf{z}) = \mathbf{0} \pmod{q}$, where $d = \gcd(\mathbf{z})$ and $g = \gcd(d, q)$. Moreover, $(g/d)\mathbf{z} \in S$ holds true and $\gcd((g/d)\mathbf{z}) = \gcd(\mathbf{z}, q)$ divides q . Let d be such that \mathcal{A} outputs, with non-negligible probability, a valid collision \mathbf{z} satisfying $\gcd(\mathbf{z}) = d$. Such a d exists because $\gcd(\mathbf{z})$ is bounded by β_∞ and divides q , so by assumption there are only polynomially many possible values of d . Let $q' = q/d$, which is an integer. By increasing m and using standard amplification techniques, we can make the probability that \mathcal{A} outputs such a collision (satisfying $\mathbf{z} \in S$, $\mathbf{A}\mathbf{z} = \mathbf{0} \pmod{q}$ and $\gcd(\mathbf{z}) = d$) exponentially close to 1.

Let (\mathbf{y}_i, s_i) for $i = 1, \dots, m$ be input samples, where \mathbf{y}_i has distribution D_{Λ, s_i} . Write each \mathbf{y}_i as $\mathbf{y}_i = \mathbf{B}\mathbf{a}_i \pmod{q'\Lambda}$ for $\mathbf{a}_i \in \mathbb{Z}_{q'}^n$. Since $s_i \geq q'\eta(\Lambda)$ the distribution of \mathbf{a}_i is statistically close to uniform over $\mathbb{Z}_{q'}^n$. Let $\mathbf{A} = [\mathbf{a}_1 \mid \dots \mid \mathbf{a}_m] \in \mathbb{Z}_q^{n \times m}$, and choose $\mathbf{A}' \in \mathbb{Z}_d^{n \times m}$ uniformly at random. Since \mathbf{A} is statistically close to uniform over $\mathbb{Z}_{q'}^{n \times m}$, the matrix $\mathbf{A} + q'\mathbf{A}'$ is statistically close to uniform over $\mathbb{Z}_q^{n \times m}$. Call the oracle \mathcal{A} on input $\mathbf{A} + q'\mathbf{A}'$, and obtain (with overwhelming probability) a nonzero $\mathbf{z} \in S$ with $\gcd(\mathbf{z}) = d$, $\|\mathbf{z}\| \leq \beta$, $\|\mathbf{z}\|_\infty \leq \beta_\infty$ and $(\mathbf{A} + q'\mathbf{A}')\mathbf{z} = \mathbf{0} \pmod{q}$. Notice that $q'\mathbf{A}'\mathbf{z} = q\mathbf{A}'(\mathbf{z}/d) = \mathbf{0} \pmod{q}$ because (\mathbf{z}/d) is an integer vector. Therefore $\mathbf{A}\mathbf{z} = \mathbf{0} \pmod{q}$. Finally, the reduction outputs (\mathbf{y}, s) , where $\mathbf{y} = \sum_i z_i \mathbf{y}_i / q$ and $s = \sqrt{\sum_i (s_i z_i)^2} / q$. Notice that $z_i \mathbf{y}_i \in q\Lambda + \mathbf{B}(z_i \mathbf{a}_i)$ because $\gcd(\mathbf{z}) = d$, so $\mathbf{y} \in \Lambda$.

Notice that s satisfies the stated bounds because \mathbf{z} is a nonzero integer vector. We next analyze the distribution of \mathbf{y} . For any fixed \mathbf{a}_i , the conditional distribution of each \mathbf{y}_i is $D_{q'\Lambda + \mathbf{B}\mathbf{a}_i, s_i}$, where $s_i \geq \sqrt{2}\eta(q'\Lambda)$. The claim on $\mathbb{E}[\|\mathbf{y}\|]$ then follows from [19, Lemma 2.11 and Lemma 4.3] and Hölder's inequality. The claim on the probability that $\mathbf{y} \notin H$ was initially shown in the preliminary version of [19]; see also [24, Lemma 3.15].

Now assume that $s_i \geq \sqrt{2}\beta_\infty q \cdot \eta(\Lambda) \geq \sqrt{2}\|\mathbf{z}\|_\infty \cdot \eta(q'\Lambda)$ for all i . By Theorem 3.3 the distribution of \mathbf{y} is statistically close to $D_{Y/q,s}$ where $Y = \gcd(\mathbf{z}) \cdot q'\Lambda + \mathbf{B}(\mathbf{A}\mathbf{z})$. Using $\mathbf{A}\mathbf{z} = \mathbf{0} \bmod q$ and $\gcd(\mathbf{z}) = d$, we get $Y = q\Lambda$. Therefore \mathbf{y} has distribution statistically close to $D_{\Lambda,s}$, as claimed. \square

Building on Lemma 3.6, our next lemma shows that for any $q \geq \beta \cdot n^{\Omega(1)}$, a collision-finding oracle can be used to obtain Gaussian samples of width close to $2\beta\beta_\infty \cdot \eta(\Lambda)$.

Lemma 3.7. *Let m, n, q, S as in Lemma 3.6, and also assume $q/\beta \geq n^\delta$ for some constant $\delta > 0$. There is an efficient reduction that, on input any basis \mathbf{B} of an n -dimensional lattice Λ , an upper bound $\eta \geq \eta(\Lambda)$, and given access to an $\text{SIS}(m, n, q, S)$ oracle (finding collisions $\mathbf{z} \in S$ with nonnegligible probability), outputs (with overwhelming probability) a sample (\mathbf{y}, s) where $\sqrt{2}\beta_\infty \cdot \eta \leq s \leq 2\beta_\infty\beta \cdot \eta$ and \mathbf{y} has distribution statistically close to $D_{\Lambda,s}$.*

Proof. By applying the LLL basis reduction algorithm [13] to the basis \mathbf{B} , we can assume without loss of generality that $\|\tilde{\mathbf{B}}\| \leq 2^n \cdot \eta(\Lambda)$. Let ω_n be an arbitrary function in n satisfying $\omega_n = \omega(\sqrt{\log n})$ and $\omega_n \leq \sqrt{n}/2$.

The main procedure, described below, produces samples having parameters in the range $[1, q] \cdot \sqrt{2}\beta_\infty \cdot \eta$. On these samples we run the procedure from Lemma 3.5 (with $R = \sqrt{2}\beta_\infty q \cdot \eta$) to obtain samples having parameters in the range $[\sqrt{2}, 2] \cdot \beta_\infty q \cdot \eta$. Finally, we invoke the reduction from Lemma 3.6 on those samples to obtain a sample satisfying the conditions in the Lemma statement.

The main procedure works in a sequence of phases $i = 0, 1, 2, \dots$. In phase i , the input is a basis \mathbf{B}_i of Λ , where initially $\mathbf{B}_0 = \mathbf{B}$. The basis \mathbf{B}_i is used in the discrete Gaussian sampling algorithm of [12, Theorem 4.1] to produce samples (\mathbf{y}, s_i) , where $s_i = \max\{\|\tilde{\mathbf{B}}_i\| \cdot \omega_n, \sqrt{2}\beta_\infty\eta\} \geq \sqrt{2}\beta_\infty\eta$ and \mathbf{y}_i has distribution statistically close to D_{Λ,s_i} . Phase i either manages to produce a sample (\mathbf{y}, s) with s in the desired range $[1, q] \cdot \sqrt{2}\beta_\infty\eta$, or it produces a new basis \mathbf{B}_{i+1} for which $\|\tilde{\mathbf{B}}_{i+1}\| \leq \|\tilde{\mathbf{B}}_i\|/2$, which is the input to the next phase. The number of phases before termination is clearly polynomial in n , by hypothesis on \mathbf{B} .

If $\|\tilde{\mathbf{B}}_i\| \cdot \omega_n \leq \sqrt{2}q\beta_\infty\eta$, then this already gives samples with $s_i \in [1, q]\sqrt{2}\beta_\infty\eta$ in the desired range, and we can terminate the main phase. So, we may assume that $s_i = \|\tilde{\mathbf{B}}_i\| \cdot \omega_n \geq \sqrt{2}q\beta_\infty\eta$. Each phase i proceeds in some constant $c \geq 1/\delta$ number of sub-phases $j = 1, 2, \dots, c$, where the inputs to the first sub-phase are the samples (\mathbf{y}, s_i) generated as described above. We recall that these samples satisfy $s_i \geq \sqrt{2}q\beta_\infty\eta$. The same will be true for the samples passed as input to all other subsequent subphases. So, each subphase receives as input samples (\mathbf{y}, s) satisfying all the hypotheses of Lemma 3.6, and we can run the reduction from that lemma to generate new samples (\mathbf{y}', s') having parameters s' bounded from above by $s_i \cdot (\beta/q)^j$, and from below by $\sqrt{2}\beta_\infty\eta$. If any of the produces samples satisfies $s' \leq q\sqrt{2}\beta_\infty\eta$, then we can terminate the main procedure with (\mathbf{y}', s') as output. Otherwise, all samples produced during the subphase satisfy $s' > q\sqrt{2}\beta_\infty\eta$, and they can be passed as input to the next sub-phase. Notice that the total runtime of all the sub-phases is $\text{poly}(n)^c$, because each invocation of the reduction from Lemma 3.6 relies on $\text{poly}(n)$ invocations of the reduction in the previous sub-phase; this is why we need to limit the number of sub-phases to a constant c .

If phase i ends up running all its sub-phases without ever finding a sample with $s' \in [1, q]\sqrt{2}\beta_\infty\eta$, then it has produced samples whose parameters are bounded by $(\beta/q)^c \leq s_i \leq s_i/\sqrt{n}$. It uses n^2 of these samples, which with overwhelming probability have lengths all bounded by s_i/\sqrt{n} , and include n linearly independent vectors. It transforms those vectors into a basis \mathbf{B}_{i+1} with $\|\tilde{\mathbf{B}}_{i+1}\| \leq s_i/\sqrt{n} \leq \|\tilde{\mathbf{B}}_i\| \cdot \omega_n/\sqrt{n} \leq \|\tilde{\mathbf{B}}_i\|/2$, as input to the next phase. \square

We can now prove our main theorem, reducing worst-case lattice problems with $\max\{1, \beta\beta_\infty/q\} \cdot \tilde{O}(\beta\sqrt{n})$ approximation factors to SIS, when $q \geq \beta \cdot n^{\Omega(1)}$.

Theorem 3.8. *Let m, n be integers, $S = \{\mathbf{z} \in \mathbb{Z}^m \setminus \{\mathbf{0}\} \mid \|\mathbf{z}\| \leq \beta \wedge \|\mathbf{z}\|_\infty \leq \beta_\infty\}$ for some real $\beta \geq \beta_\infty > 0$, and $q \geq \beta \cdot n^{\Omega(1)}$ be an integer modulus with at most $\text{poly}(n)$ integer divisors less than β_∞ . For some $\gamma = \max\{1, \beta\beta_\infty/q\} \cdot O(\beta\sqrt{n})$, there is an efficient reduction from SIVP_γ^η (and hence also from standard $\text{SIVP}_{\gamma\omega_n}$) on n -dimensional lattices to S -collision finding for $\text{SIS}(m, n, q)$ with non-negligible advantage.*

Proof. Given an input basis \mathbf{B} of a lattice Λ , we can apply the LLL algorithm to obtain a 2^n -approximation to $\eta(\Lambda)$, and by scaling we can assume that $\eta(\Lambda) \in [1, 2^n]$. For $i = 1, \dots, n$, we run the procedure described below for each hypothesized upper bound $\eta_i = 2^i$ on $\eta(\Lambda)$. Each call to the procedure either fails, or returns a set of linearly independent vectors in Λ whose lengths are all bounded by $(\gamma/2) \cdot \eta_i$. We return the first such obtained set (i.e., for the minimal value of i). As we show below, as long as $\eta_i \geq \eta(\Lambda)$ the procedure returns a set of vectors with overwhelming probability. Since one $\eta_i \in [1, 2) \cdot \eta(\Lambda)$, our reduction solves SIVP_γ^η with overwhelming probability, as claimed.

The procedure invokes the reduction from Lemma 3.7 with $\eta = \eta_i$ to obtain samples with parameters in the range $[\sqrt{2}\beta_\infty, \sqrt{2}\beta\beta_\infty] \cdot \eta$. On these samples we run the procedure from Lemma 3.5 with $R = \max\{\sqrt{2}q, \sqrt{2}\beta\beta_\infty\}$ to obtain samples having parameters in the range $[R, \sqrt{2}R] \cdot \eta$. On such samples we repeatedly run (using independent samples each time) the reduction from Lemma 3.6. After enough runs, we obtain with overwhelming probability a set of linearly independent lattice vectors all having lengths at most $(\gamma/2) \cdot \eta$, as required. \square

4 Hardness of LWE with Small Uniform Errors

In this section we prove the hardness of inverting the LWE function even when the error vectors have very small entries, provided the number of samples is sufficiently small. We proceed similarly to [23, 4], by using the LWE assumption (for discrete Gaussian error) to construct a lossy family of functions with respect to a uniform distribution over small inputs. However, the parameterization we obtain is different from those in [23, 4], allowing us to obtain *pseudorandomness* of LWE under *very small* (e.g., binary) inputs, for a number of LWE samples that exceeds the LWE dimension.

Our results and proofs are more naturally formulated using the SIS function family. So, we will first study the problem in terms of SIS, and then reformulate the results in terms of LWE using Proposition 2.9. We recall that the main difference between this section and Section 3, is that here we consider parameters for which the resulting functions are essentially injective, or more formally, statistically second-preimage resistant. The following lemma gives sufficient conditions that ensure this property.

Lemma 4.1. *For any integers m, k, q, s and set $X \subseteq [s]^m$, the function family $\text{SIS}(m, k, q)$ is (statistically) ϵ -second preimage resistant with respect to the uniform input distribution $\mathcal{U}(X)$ for $\epsilon = |X| \cdot (s'/q)^k$, where s' is the largest factor of q smaller than s .*

Proof. Let $\mathbf{x} \leftarrow \mathcal{U}(X)$ and $\mathbf{A} \leftarrow \text{SIS}(m, k, q)$ be chosen at random. We want to evaluate the probability that there exists an $\mathbf{x}' \in X \setminus \{\mathbf{x}\}$ such that $\mathbf{A}\mathbf{x} = \mathbf{A}\mathbf{x}' \pmod{q}$, or, equivalently, $\mathbf{A}(\mathbf{x} - \mathbf{x}') = \mathbf{0} \pmod{q}$. Fix any two distinct vectors $\mathbf{x}, \mathbf{x}' \in X$ and let $\mathbf{z} = \mathbf{x} - \mathbf{x}'$. The vector $\mathbf{A}\mathbf{z} \pmod{q}$ is distributed uniformly at random in $(d\mathbb{Z}/q\mathbb{Z})^k$, where $d = \gcd(q, z_1, \dots, z_m)$. All coordinates of \mathbf{z} are in the range $z_i \in \{-(s-1), \dots, (s-1)\}$, and at least one of them is nonzero. Therefore, d is at most s' and $|d\mathbb{Z}_q^k| = (q/d)^k \geq (q/s')^k$. By union bound (over $\mathbf{x}' \in X \setminus \{\mathbf{x}\}$) for any \mathbf{x} , the probability that there is a second preimage \mathbf{x}' is at most $(|X| - 1)(s'/q)^k$. \square

We remark that, as shown in Section 3, even for parameter settings that do not fall within the range specified in Lemma 4.1, $\text{SIS}(m, k, q)$ is collision resistant, and therefore also (computationally) second-preimage-resistant. This is all that is needed in the rest of this section. However, when $\text{SIS}(m, k, q)$ is not statistically second-preimage resistant, the one-wayness proof that follows (see Theorem 4.5) is not very interesting: typically, in such settings, $\text{SIS}(m, k, q)$ is also statistically uninvertible, and the one-wayness of $\text{SIS}(m, k, q)$ directly follows from Lemma 2.2. So, below we focus on parameter settings covered by Lemma 4.1.

We prove the one-wayness of $\mathcal{F} = \text{SIS}(m, k, q, X)$ with respect to the uniform input distribution $\mathcal{X} = \mathcal{U}(X)$ by building a lossy function family $(\mathcal{L}, \mathcal{F}, \mathcal{X})$ where \mathcal{L} is an auxiliary function family that we will prove to be uninvertible and computationally indistinguishable from \mathcal{F} . The auxiliary family \mathcal{L} is derived from the following function family.

Definition 4.2. For any probability distribution \mathcal{Y} over \mathbb{Z}^ℓ and integer $m \geq \ell$, let $\mathcal{I}(m, \ell, \mathcal{Y})$ be the probability distribution over linear functions $[\mathbf{I} \mid \mathbf{Y}]: \mathbb{Z}^m \rightarrow \mathbb{Z}^\ell$ where \mathbf{I} is the $\ell \times \ell$ identity matrix, and $\mathbf{Y} \in \mathbb{Z}^{\ell \times (m-\ell)}$ is obtained choosing each column of \mathbf{Y} independently at random from \mathcal{Y} .

We anticipate that we will set \mathcal{Y} to the Gaussian input distribution $\mathcal{Y} = D_{\mathbb{Z}, \sigma}^\ell$ in order to make \mathcal{L} indistinguishable from \mathcal{F} under a standard LWE assumption. But for generality, we prove some of our results with respect to a generic distribution \mathcal{Y} .

The following lemma shows that for a bounded distribution \mathcal{Y} (and appropriate parameters), $\mathcal{I}(m, \ell, \mathcal{Y})$ is (statistically) uninvertible.

Lemma 4.3. Let \mathcal{Y} be a probability distribution on $[\mathcal{Y}] \subseteq \{-\sigma, \dots, \sigma\}^n$, and let $X \subseteq \{-s, \dots, s\}^m$. Then $\mathcal{I}(m, \ell, \mathcal{Y})$ is ϵ -uninvertible with respect to $\mathcal{U}(X)$ for $\epsilon = (1 + 2s(1 + \sigma(m - \ell)))^\ell / |X|$.

Proof. Let $f = [\mathbf{I} \mid \mathbf{Y}]$ be an arbitrary function in the support of $\mathcal{I}(m, \ell, \mathcal{Y})$. We know that $|y_{i,j}| \leq \sigma$ for all i, j . We first bound the size of the image $|f(X)|$. By the triangle inequality, all the points in the image $f(X)$ have ℓ_∞ norm at most

$$\|f(\mathbf{u})\|_\infty \leq \|\mathbf{u}\|_\infty(1 + \sigma(m - \ell)) \leq s(1 + \sigma(m - \ell)).$$

The number of integer vectors (in \mathbb{Z}^ℓ) with such bounded ℓ_∞ norm is

$$(1 + 2s(1 + \sigma(m - \ell)))^\ell.$$

Dividing by the size of X and using Lemma 2.4, the claim follows. \square

Lemma 4.3 applies to any distribution \mathcal{Y} with bounded support. When $\mathcal{Y} = D_{\mathbb{Z}, \sigma}^\ell$ is a discrete Gaussian distribution, a slightly better bound can be obtained. (See also [4], which proves a similar lemma for a different, non-uniform input distribution \mathcal{X} .)

Lemma 4.4. Let $\mathcal{Y} = D_{\mathbb{Z}, \sigma}^\ell$ be the discrete Gaussian distribution with parameter $\sigma > 0$, and let $X \subseteq \{-s, \dots, s\}^m$. Then $\mathcal{I}(m, \ell, \mathcal{Y})$ is ϵ -uninvertible with respect to $\mathcal{U}(X)$, for $\epsilon = O(\sigma m s / \sqrt{\ell})^\ell / |X| + 2^{-\Omega(m)}$.

Proof. Again, by Lemma 2.4 it is enough to bound the expected size of $f(X)$ when $f \leftarrow \mathcal{I}(m, \ell, \mathcal{Y})$ is chosen at random. Remember that $f = [\mathbf{I} \mid \mathbf{Y}]$ where $\mathbf{Y} \leftarrow D_{\mathbb{Z}, \sigma}^{\ell \times (m-\ell)}$. Since the entries of $\mathbf{Y} \in \mathbb{R}^{\ell \times (m-\ell)}$ are independent mean-zero subgaussians with parameter σ , by a standard bound from the theory of random matrices, the largest singular value $s_1(\mathbf{Y}) = \max_{\mathbf{0} \neq \mathbf{x} \in \mathbb{R}^m} \|\mathbf{Y}\mathbf{x}\| / \|\mathbf{x}\|$ of \mathbf{Y} is at most $\sigma \cdot O(\sqrt{\ell} + \sqrt{m - \ell}) =$

$\sigma \cdot O(\sqrt{m})$, except with probability $2^{-\Omega(m)}$. We now bound the ℓ_2 norm of all vectors in the image $f(X)$. Let $\mathbf{u} = (\mathbf{u}_1, \mathbf{u}_2) \in X$, with $\mathbf{u}_1 \in \mathbb{Z}^\ell$ and $\mathbf{u}_2 \in \mathbb{Z}^{m-\ell}$. Then

$$\begin{aligned}
\|f(\mathbf{u})\| &\leq \|\mathbf{u}_1 + \mathbf{Y}\mathbf{u}_2\| \\
&\leq \|\mathbf{u}_1\| + \|\mathbf{Y}\mathbf{u}_2\| \\
&\leq \left(\sqrt{\ell} + s_1(\mathbf{Y})\sqrt{m-\ell}\right) s \\
&\leq \left(\sqrt{\ell} + \sigma \cdot O(\sqrt{m})\sqrt{m-\ell}\right) s \\
&= O(\sigma m s).
\end{aligned}$$

The number of integer points in the ℓ -dimensional zero-centered ball of radius $R = O(\sigma m s)$ can be bounded by a simple volume argument, as $|f(X)| \leq (R + \sqrt{\ell}/2)^n V_\ell = O(\sigma m s / \sqrt{\ell})^\ell$, where $V_\ell = \pi^{\ell/2}/(\ell/2)!$ is the volume of the ℓ -dimensional unit ball. Dividing by the size of X and accounting for the rare event that $s_1(\mathbf{Y})$ is not bounded as above, we get that $\mathcal{I}(m, \ell, \mathcal{Y})$ is ϵ -uninvertible for $\epsilon = O(\sigma m s / \sqrt{\ell})^\ell / |X| + 2^{-\Omega(m)}$. \square

We can now prove the one-wayness of the SIS function family by defining and analyzing an appropriate lossy function family. The parameters below are set up to expose the connection with LWE, via Proposition 2.9: $\text{SIS}(m, m-n, q)$ corresponds to LWE in n dimensions (given m samples), whose one-wayness we are proving, while $\text{SIS}(\ell = m-n+k, m-n, q)$ corresponds to LWE in $k \leq n$ dimensions, whose pseudorandomness we are assuming.

Theorem 4.5. *Let q be a modulus and let \mathcal{X}, \mathcal{Y} be two distributions over \mathbb{Z}^m and \mathbb{Z}^ℓ respectively, where $\ell = m-n+k$ for some $0 < k \leq n \leq m$, such that*

1. $\mathcal{I}(m, \ell, \mathcal{Y})$ is uninvertible with respect to input distribution \mathcal{X} ,
2. $\text{SIS}(\ell, m-n, q)$ is pseudorandom with respect to input distribution \mathcal{Y} , and
3. $\text{SIS}(m, m-n, q)$ is second-preimage resistant with respect to input distribution \mathcal{X} .

Then $\mathcal{F} = \text{SIS}(m, m-n, q)$ is one-way with respect to input distribution \mathcal{X} .

In particular, if $\text{SIS}(\ell, m-n, q)$ is pseudorandom with respect to the discrete Gaussian distribution $\mathcal{Y} = D_{\mathbb{Z}, \sigma}^\ell$, then $\text{SIS}(m, m-n, q)$ is $(2\epsilon + 2^{-\Omega(m)})$ -one-way with respect to the uniform input distribution $\mathcal{X} = \mathcal{U}(X)$ over any set $X \subseteq \{-s, \dots, s\}^m$ satisfying

$$(C' \sigma m s / \sqrt{\ell})^\ell / \epsilon \leq |X| \leq \epsilon \cdot (q/s')^{m-n},$$

where s' is the largest divisor of q that is smaller than or equal to $2s$, and C' is the universal constant hidden by the $O(\cdot)$ notation from Lemma 4.4.

Proof. We will prove that $(\mathcal{L}, \mathcal{F}, \mathcal{X})$ is a lossy function family, where $\mathcal{F} = \text{SIS}(m, m-n, q)$ and $\mathcal{L} = \text{SIS}(\ell, m-n, q) \circ \mathcal{I}(m, \ell, \mathcal{Y})$. It follows from Lemma 2.3 that both \mathcal{F} and \mathcal{L} are one-way function families with respect to input distribution \mathcal{X} . Notice that \mathcal{F} is second-preimage resistant with respect to \mathcal{X} by assumption. The indistinguishability of \mathcal{L} and \mathcal{F} follows immediately from the pseudorandomness of $\text{SIS}(\ell, m-n, q)$ with respect to \mathcal{Y} , by a standard hybrid argument. So, in order to prove that $(\mathcal{L}, \mathcal{F}, \mathcal{X})$ is a lossy function family, it suffices to prove that \mathcal{L} is uninvertible with respect to \mathcal{X} . This follows applying Lemma 2.5 to the function family $\mathcal{I}(m, \ell, \mathcal{Y})$, which is uninvertible by assumption. This proves the first part of the theorem.

Now consider the particular instantiation. Let $\mathcal{X} = \mathcal{U}(X)$ be the uniform distribution over a set $X \subseteq \{-s, \dots, s\}^m$ whose size satisfies the inequalities in the theorem statement, and let $\mathcal{Y} = D_{\mathbb{Z}, \sigma}^\ell$. Since $|X|(s'/q)^{m-n} \leq \epsilon$, by Lemma 4.1, $\text{SIS}(m, m-n, q)$ is (statistically) second-preimage resistant with respect to input distribution \mathcal{X} . Moreover, since $(C\sigma ms/\sqrt{\ell})^\ell/|X| \leq \epsilon$, by Lemma 4.4, $\mathcal{I}(m, \ell, \mathcal{Y})$ is $(\epsilon + 2^{-\Omega(m)})$ -uninvertible with respect to input distribution \mathcal{X} . \square

In order to conclude that the LWE function is pseudorandom (under worst-case lattice assumptions) for uniformly random small errors, we combine Theorem 4.5 with Corollary 2.14, instantiating the parameters appropriately. For simplicity, we focus on the important case of a prime modulus q . Nearly identical results for composite moduli (e.g., those divisible by only small primes) are also easily obtained from Corollary 2.14, or by using either Theorem 2.13 or Theorem 2.12.

Theorem 4.6. *Let $0 < k \leq n \leq m - \omega(\log k) \leq k^{O(1)}$, $\ell = m - n + k$, $s \geq (Cm)^{\ell/(n-k)}$ for a large enough universal constant C , and q be a prime such that $\max\{3\sqrt{k}, (4s)^{m/(m-n)}\} \leq q \leq k^{O(1)}$. For any set $X \subseteq \{-s, \dots, s\}^m$ of size $|X| \geq s^m$, the $\text{SIS}(m, m-n, q)$ (equivalently, $\text{LWE}(m, n, q)$) function family is one-way (and pseudorandom) with respect to the uniform input distribution $\mathcal{X} = \mathcal{U}(X)$, under the assumption that SIVP_γ is (quantum) hard to approximate, in the worst case, on k -dimensional lattices to within a factor $\gamma = \tilde{O}(\sqrt{k} \cdot q)$.*

A few notable instantiations are as follows. To obtain pseudorandomness for binary errors, we need $s = 2$ and $X = \{0, 1\}^m$. For this value of s , the condition $s \geq (Cm)^{\ell/(n-k)}$ can be equivalently be rewritten as

$$m \leq (n - k) \cdot \left(1 + \frac{1}{\log_2(Cm)}\right),$$

which can be satisfied by taking $k = n/(C' \log_2 n)$ and $m = n(1 + 1/(c \log_2 n))$ for any desired $c > 1$ and a sufficiently large constant $C' > 1/(1 - 1/c)$. For these values, the modulus should satisfy $q \geq 8^{m/(m-n)} = 8n^{3c} = k^{O(1)}$, and can be set to any sufficiently large prime $p = k^{O(1)}$.¹

Notice that for binary errors, both the worst-case lattice dimension k and the number $m - n$ of “extra” LWE samples (i.e., the number of samples beyond the LWE dimension n) are both sublinear in the LWE dimension n : we have $k = \Theta(n/\log n)$ and $m - n = O(n/\log n)$. This corresponds to both a stronger worst-case security assumption, and a less useful LWE problem. By using larger errors, say, bounded by $s = n^\epsilon$ for some constant $\epsilon > 0$, it is possible to make both the worst-case lattice dimension k and number of extra samples $m - n$ into (small) linear functions of the LWE dimension n , which may be sufficient for some cryptographic applications of LWE. Specifically, for any constant $\epsilon < 1$, one may set $k = (\epsilon/3)n$ and $m = (1 + \epsilon/3)n$, which are easily verified to satisfy all the hypotheses of Theorem 4.6 when $q = k^{O(1)}$ is sufficiently large. These parameters correspond to $(\epsilon/3)n = \Omega(n)$ extra samples (beyond the LWE dimension n), and to the worst-case hardness of lattice problems in dimension $(\epsilon/3)n = \Omega(n)$. Notice that for $\epsilon < 1/2$, this version of LWE has much smaller errors than allowed by previous LWE hardness proofs, and it would be subject to subexponential-time attacks [2] if the number of samples were not restricted. Our result shows that if the number of samples is limited to $(1 + \epsilon/3)n$, then LWE maintains its provable security properties and conjectured exponential-time hardness in the dimension n .

One last instantiation allows for a linear number of samples $m = c \cdot n$ for any desired constant $c \geq 1$, which is enough for most applications of LWE in lattice cryptography. In this case we can choose (say)

¹Here we have not tried to optimize the value of q , and smaller values of the modulus are certainly possible: a close inspection of the proof of Theorem 4.6 reveals that for binary errors, the condition $q \geq 8n^{3c}$ can be replaced by $q \geq n^{c'}$ for any constant $c' > c$.

$k = n/2$, and it suffices to set the other parameters so that

$$s \geq (Cm)^{2c-1} \quad \text{and} \quad q \geq (4s)^{c/(c-1)} \geq 4^{c/(c-1)} \cdot (Ccn)^{2c+1+1/(c-1)} = k^{O(1)}.$$

(We can also obtain better lower bounds on s and q by letting k be a smaller constant fraction of n .) This proves the hardness of LWE with uniform noise of polynomial magnitude $s = n^{O(1)}$, and any linear number of samples $m = O(n)$. Note that for $m = cn$, any instantiation of the parameters requires the magnitude s of the errors to be at least n^{c-1} . For $c > 3/2$, this is more noise than is typically used in the standard LWE problem, which allows errors of magnitude as small as $O(\sqrt{n})$, but requires them to be independent and follow a Gaussian-like distribution. The novelty in this last instantiation of Theorem 4.6 is that it allows for a much wider class of error distributions, including the uniform distribution, and distributions where different components of the error vector are correlated.

Proof of Theorem 4.6. We prove the one-wayness of $\text{SIS}(m, m - n, q)$ (equivalently, $\text{LWE}(m, n, q)$ via Proposition 2.9) using the second part of Theorem 4.5 with $\sigma = 3\sqrt{k}$. Using $\ell \geq k$ and the primality of q , the conditions on the size of X in Theorem 4.5 can be replaced by simpler bounds

$$\frac{(3C'ms)^\ell}{\epsilon} \leq |X| \leq \epsilon \cdot q^{m-n},$$

or equivalently, the requirement that the quantities $(3C'ms)^\ell/|X|$ and $|X|/q^{m-n}$ are negligible in k . For the first quantity, letting $C = 4C'$ and using $|X| \geq s^m$ and $s \geq (4C'm)^{\ell/(n-k)}$, we get that $(3C'ms)^\ell/|X| \leq (3/4)^{-\ell} \leq (3/4)^{-k}$ is exponentially small (in k). For the second quantity, using $|X| \leq (2s + 1)^m$ and $q \geq (4s)^{m/(m-n)}$, we get that $|X|/q^{m-n} \leq (3/4)^m$ is also exponentially small.

Theorem 4.5 also requires the pseudorandomness of $\text{SIS}(\ell, m - n, q)$ with respect to the discrete Gaussian input distribution $\mathcal{Y} = D_{\mathbb{Z}, \sigma}^\ell$, which can be based on the (quantum) worst-case hardness of SIVP on k -dimensional lattices using Corollary 2.14. (Notice the use of different parameters: $\text{SIS}(m, m - n, q)$ in Corollary 2.14, and $\text{SIS}(m - n + k, m - n, q)$ here.) After properly renaming the variables, and using $\sigma = 3\sqrt{k}$, the hypotheses of Corollary 2.14 become $\omega(\log k) \leq m - n \leq k^{O(1)}$, $3\sqrt{k} < q < k^{O(1)}$, which are all satisfied by the hypotheses of the Theorem. The corresponding assumption is the worst-case hardness of SIVP_γ on k -dimensional lattices, for $\gamma = k\omega_k q/\sigma = \sqrt{k}\omega_k q/3 = \tilde{O}(\sqrt{k}q)$, as claimed. This concludes the proof of the one-wayness of LWE.

The pseudorandomness of LWE follows from the sample-preserving search-to-decision reduction of [17]. \square

References

- [1] M. Ajtai. Generating hard instances of lattice problems. *Quaderni di Matematica*, 13:1–32, 2004. Preliminary version in STOC 1996.
- [2] S. Arora and R. Ge. New algorithms for learning in presence of errors. In *ICALP (I)*, pages 403–415, 2011.
- [3] A. Banerjee, C. Peikert, and A. Rosen. Pseudorandom functions and lattices. In *EUROCRYPT*, pages 719–737, 2012.
- [4] M. Bellare, E. Kiltz, C. Peikert, and B. Waters. Identity-based (lossy) trapdoor functions and applications. In *EUROCRYPT*, pages 228–245, 2012.

- [5] A. Blum, A. Kalai, and H. Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. *J. ACM*, 50(4):506–519, 2003.
- [6] D. Boneh and D. M. Freeman. Linearly homomorphic signatures over binary fields and new tools for lattice-based signatures. In *Public Key Cryptography*, pages 1–16, 2011.
- [7] J.-Y. Cai and A. Nerurkar. An improved worst-case to average-case connection for lattice problems. In *FOCS*, pages 468–477, 1997.
- [8] Y. Chen and P. Q. Nguyen. BKZ 2.0: Better lattice security estimates. In *ASIACRYPT*, pages 1–20, 2011.
- [9] D. Dadush, C. Peikert, and S. Vempala. Enumerative lattice algorithms in any norm via M-ellipsoid coverings. In *FOCS*, pages 580–589, 2011.
- [10] N. Döttling and J. Müller-Quade. Lossy codes and a new variant of the learning-with-errors problem. Manuscript. To appear in Eurocrypt 2013, 2013.
- [11] N. Gama and P. Q. Nguyen. Predicting lattice reduction. In *EUROCRYPT*, pages 31–51, 2008.
- [12] C. Gentry, C. Peikert, and V. Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC*, pages 197–206, 2008.
- [13] A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, December 1982.
- [14] D. Micciancio. Almost perfect lattices, the covering radius problem, and applications to Ajtai’s connection factor. *SIAM J. Comput.*, 34(1):118–169, 2004.
- [15] D. Micciancio. Duality in lattice cryptography. In *Public Key Cryptography*, 2010. Invited talk.
- [16] D. Micciancio and S. Goldwasser. *Complexity of Lattice Problems: a cryptographic perspective*, volume 671 of *The Kluwer International Series in Engineering and Computer Science*. Kluwer Academic Publishers, Boston, Massachusetts, 2002.
- [17] D. Micciancio and P. Mol. Pseudorandom knapsacks and the sample complexity of LWE search-to-decision reductions. In *CRYPTO*, pages 465–484, 2011.
- [18] D. Micciancio and C. Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *EUROCRYPT*, pages 700–718, 2012.
- [19] D. Micciancio and O. Regev. Worst-case to average-case reductions based on Gaussian measures. *SIAM J. Comput.*, 37(1):267–302, 2007. Preliminary version in FOCS 2004.
- [20] D. Micciancio and O. Regev. Lattice-based cryptography. In *Post Quantum Cryptography*, pages 147–191. Springer, February 2009.
- [21] C. Peikert. Public-key cryptosystems from the worst-case shortest vector problem. In *STOC*, pages 333–342, 2009.
- [22] C. Peikert. An efficient and parallel Gaussian sampler for lattices. In *CRYPTO*, pages 80–97, 2010.

- [23] C. Peikert and B. Waters. Lossy trapdoor functions and their applications. In *STOC*, pages 187–196, 2008.
- [24] O. Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6):1–40, 2009. Preliminary version in *STOC* 2005.
- [25] D. Wagner. A generalized birthday problem. In *CRYPTO*, pages 288–303, 2002.

How to Delegate Secure Multiparty Computation to the Cloud

Abstract

We study the problem of verifiable computation in the presence of many clients who rely on a server to perform computations over inputs privately held by each client. This generalizes the single-client model for verifiable outsourced computation previously studied in the literature.

We put forward a computational model and strong simulation-based security for this task. We then present a new protocol that allow the clients to securely outsource an arbitrary polynomial-time computation over privately held inputs to a powerful server. At the end, the clients will be assured that the result of the computation is correct, while at the same time protecting their data from the server and each other.

Our new protocol satisfies the crucial efficiency requirement of outsourced computation where the work of the client is substantially smaller than what is required to compute the function. We use the Gennaro *et al.* amortized model, where the clients are allowed to invest in a one-time computationally expensive preprocessing phase. Our protocol is secure in the real/ideal paradigm even when dishonest clients can collude with the server in order to learn honest party's inputs or in order to maliciously change the output of the computation.

Keywords: verifiable computation, secure multi-party computation

1 Introduction

Consider the following scenario: a set of computationally weak devices holding private inputs, wish to jointly compute a function F over those inputs. Each device does not have the power to compute F on its own, let alone engaging in a secure multiparty computation protocol such as [Yao82, GMW87, BOGW88, CCD88] for computing F . They can however access the services of a “computation provider” who can “help” them compute F . To maintain the privacy of their input, the clients need to engage in a cryptographic protocol where the provider does the bulk of the computation (i.e., computes F), while the computation and communication of each client is “low” (in particular, less than the time it takes to compute F). Nevertheless the clients must be able to verify the correctness of the output of this protocol, even under the assumption that some corrupted clients might cooperate with a malicious provider to fool them into accepting an incorrect output or to learn their input.

One can think of this problem, called *multi-client verifiable computation* or *multi-client delegation of computation*, as a secure multi-party computation protocol between the clients and the provider (the cloud service), where however only the provider’s work is allowed to be proportional to the complexity of the function being computed (the function that computes the joint statistics). In this paper, we solve the above problem, by relying on only *standard polynomial time cryptographic assumptions*. We present a protocol that allows many computationally weak clients to securely outsource a computation over privately held inputs to a powerful server, in the presence of the most powerful adversarial model that can be considered, and by minimizing the “on-line” computation by the clients. The round complexity being of paramount importance in this line of work, we follow the convention of obtaining a solution in which the clients delegates the computation non-interactively (i.e., the clients and the server exchange a single message).

While a lot of work has been devoted to secure outsourced computation in the case of a single client interacting with a single server (see for example [Mic94, GKR08, GGP10, CKV10, AIK10]), the research effort for the multi-client case is still in the preliminary stages with very few works that consider much weaker models of security (we shall discuss these works in detail later on).

1.1 Our Model

Before we state our main results, we shall first take a closer look at the model in which we work in. In a nutshell, we obtain our results a) based on standard cryptographic hardness assumptions; b) in the strongest adversarial model - i.e., simulation based security in the ideal/real paradigm when malicious clients may collude with the server; and c) with minimal communication between the clients and that too only when verifying the results of the computation. In explaining our model, we consider three important design principles that influence our choice - first, the cryptographic hardness assumption that we make; second, the corruption model (which parties can the adversary corrupt), and finally, the communication model (how much do clients need to interact with each other and with the server).

Hardness assumptions: *standard cryptographic assumptions.* Following the standard convention in cryptography, we are interested in constructing multi-client verifiable computation protocols based on *standard* cryptographic assumptions (i.e., without resorting to random oracles or non-falsifiable hardness assumptions). Furthermore, we are interested in obtaining solutions where the interaction between the clients and the server is minimal, i.e., only one message is sent in each direction between the client and the server. We note that obtaining such solutions is a difficult problem even in the single-client setting, exemplified by the small number of known solutions [GGP10, CKV10, AIK10]. In particular, all known single-client non-interactive solutions based on standard assumptions work in an “amortized” computational model (also known as the pre-processing model) [GGP10]. In view of the above, in this work, we will also work in (a natural extension of) the pre-processing model, which we discuss later on in this section.

Corruption model: *simulation based security.* As we discussed earlier, it is quite natural to have a situation where one of the clients might collude with a corrupt server in order to either learn something about the honest client’s inputs or to force the output of the computation to some value. Naturally, it would be highly desirable to

construct protocols that are secure even against such adversaries. Furthermore, simulation-based security in the real/ideal security paradigm being the benchmark for security in cryptography, we would like to obtain protocols that are secure in this sense. Finally, we would like our protocols to be as widely applicable as possible, thus we choose to work in the strongest corruption model.

We remark that the outsourcing of multi-party computation has been studied in weaker security models [KMR11, CKKC13]. We shall discuss these works more in detail later on in this section.

Communication model: *minimal interaction between parties.* Since we wish to only rely on standard cryptographic assumptions, we shall work in the pre-processing model. In the pre-processing model, the clients, at the start of the protocol, execute a *preprocessing phase* which is a one-time stage in which the clients compute public as well as private information associated with the function F that they wish to outsource. The computation of each client in this phase is allowed to be proportional to the computational complexity of evaluating F . Communication being at a premium, one would like to have protocols in which the interaction between the clients and the server (and amongst the clients) is minimized. Now, ideally, it would be great if one could obtain a protocol in which the clients interacted only in the pre-processing phase, then interacted with the server once individually (by sending and receiving exactly one message) and then obtained the results of the computation. Unfortunately, this is impossible to achieve in our security model - one can easily see that if the clients did not interact with each other, after exchanging one message with the server, then one cannot obtain a simulation-based secure protocol that is secure against a colluding client and server (more specifically, for a fixed input of the honest client, the colluding client and server would be able to obtain the output of the computation on several inputs of their choice, thus violating the requirements of a simulation-based definition). Thus, clients need to interact with each other in order to obtain the results of the computation; the focus would then be on minimizing this interaction.

The above choices (allowing the clients to perform expensive computation and communication during the off-line phase, but then restricting them to a single message exchange during the on-line phase) might seem artificial. Yet there are several practical scenarios where this are relevant. Consider the case of military coalitions where the clients are armies from different countries and are in need to perform joint computations on data that might need to be kept private by each army. It is conceivable that the off-line phase will be performed over a trusted network before the deployment of soldiers in the field, and therefore computation and communication are not at a premium. The situation however changes dramatically during the on-line phase where the input to the computation is obtained during actual combat operations where battery power and communication bandwidth might be severely limited.

Advantages of the communication model. Our communication model has two further advantages:

- The foremost advantage of our communication model is that of *asynchronicity*. Note that during the outsourcing of computation, none of the clients need be present at the same time. They can send their respective messages to the servers at various points of time. Only when they wish to verify the computation do clients have to synchronize and run a computation (which is unavoidable within our framework of security).
- Another advantage of the clients not communicating during the online phase is that clients could *batch* together multiple computations and at the end could verify all the computations together.

Description of our model. Given the most natural and useful design choices above, we now describe the model in which we obtain our protocols. Our protocol for outsourcing multi-party computation consists of three phases: the preprocessing phase, the online phase, and the offline phase.

The *preprocessing phase* is a one-time stage in which the clients compute public as well as private information associated with the function F that they wish to outsource. The computation of each client in this phase is allowed to be proportional to the computational complexity of evaluating F . We also allow clients to interact with each other in an arbitrary manner in this phase. This phase is executed only once and is independent of the client's inputs.

In the *online phase* clients *individually* prepare public and private information about their respective inputs and send a single message to the server. The server, upon receiving these messages from all the clients, sends back a single message to each of the clients (the messages sent by the server to each of the clients can potentially be different from one another). Note that we do not allow the clients to communicate directly with each other in this phase. The computational complexity of the clients in this phase is required to be independent of F .

Finally, in the *offline phase*, the clients interact with each other to decode the response provided to them by the server and obtain the result of the computation. We will focus on minimizing the interaction between the clients in this phase. Furthermore, the computational complexity of the clients in this phase is also required to be independent of F .

We obtain a protocol that enjoys simulation-based security in the real/ideal paradigm and is secure even against an adversarial client who colludes with a malicious server. As we will show, requiring security against a colluding adversary and requiring a simulation based definition of security to be met, present significant challenges that need to be overcome in order to construct a protocol for outsourcing multi-party computation. Finally, our protocol makes use of standard cryptographic assumptions (and not random oracles or non-falsifiable assumptions). We note here that our solution, just like the solutions for single-party verifiable computation [GGP10, CKV10], require that the pre-processing phase be executed again, in the event that the output of a computation is rejected by one of the clients. In other words, we cannot reveal to a malicious server that the result of the computation was rejected, and then continue with another verifiable computation protocol with the same pre-processing information.

Alternative approaches to delegating multi-party computation. Note, that if one were to resort to using random oracles or making use of non-falsifiable hardness assumptions [Nao03], then it is easy to construct multi-client verifiable computation protocols. Very briefly, the clients can simply send their inputs to the server and the server can return the result of the computation along with a succinct non-interactive argument (SNARG) [Mic94, GW11, BCCT12, GLR11, DFH12] proving that it evaluated the output honestly. Privacy of the clients inputs can be obtained through standard techniques (e.g., via the use of fully homomorphic encryption). However, this solution is uninteresting from the point of view of the non-standard hardness assumption required to prove it secure.

Also if we relax the security notion to only consider non-colluding adversaries (that is a malicious client and a malicious server do not collude), and if we do not wish to obtain the stronger simulation-based definition of security, then the work of Kamara, Mohassel, and Raykova [KMR11] shows how to outsource multi-party computation. With the important focus on removing interaction between clients, the work of Choi *et al.* [CKKC13] consider multi-client non-interactive verifiable computation in which soundness guarantees are provided against a malicious server when all clients are honest; they also define privacy guarantees separately against a server and against a client. We note that this is much weaker than the simulation based security model that we work in that captures soundness and privacy against malicious clients and server colluding with each other.

Finally, we remark that if we did allow the clients to interact in the online phase (and sacrifice on asynchronicity), then one can trivially obtain a protocol for outsourcing multi-party computation from any single-party protocol for outsourcing computation [GGP10, CKV10, AIK10]; in the online phase, the clients simply “simulate” a single party by running a secure computation protocol to compute the message sent by the client in the single-party protocol. As discussed above, in our view, this is a particularly unsatisfactory approach, and of limited interest.

1.2 Our Results

In this work, we show how to construct a secure protocol for two-party verifiable computation in the pre-processing model. We highlight the key features in our protocol:

- In our solution, the clients perform work proportional to F only in the pre-processing phase (executed once), and have computational complexity independent of F in the remainder of the protocol (the online

phase and the offline phase).

- The clients exchange only a single message with the server (in the online phase) and hence our protocols are “non-interactive” with regards to interaction with the server.
- Furthermore, a critical point of our protocol is that the clients do not have to interact with each other in the online phase - this allows the clients to batch together several (unbounded) computations together before running the verification to obtain the results of the computations.
- We provide simulation-based security via the real/ideal security paradigm and our protocol is secure against a colluding adversarial client and adversarial server. This provides both soundness of the computation (an honest client never accepts an incorrect output) and privacy of the honest client’s inputs.

We then show how to extend our two-party protocol to the multi-party setting and obtain a multi-party outsourcing computation protocol with the same features as above (our protocol is secure against a constant fraction of adversarial clients who may collude with a malicious server). Finally, we show how to reduce the interaction between the clients in the offline phase.

In other words, our protocol works in the strongest possible adversarial model, and apart from the preprocessing phase, has minimal interaction.

1.3 Overview

Starting point. We start with the goal of trying to “bootstrap” a single-client delegation scheme into a delegation scheme for multiple clients. Thus, our first consideration is what kind of properties are desirable from a single client delegation scheme in order to achieve our goal. Interestingly, we observe that constructing a multi-client delegation scheme against *colluding* adversaries turns out to be very sensitive to the choice of the underlying single-client delegation scheme. Specifically, recall that the construction of [GGP10] uses the authenticity property of Yao’s garbled circuits to enforce correctness of computation. However, analyzing the security of garbled circuits in the setting where the honest client may share secret keys with the adversary (which seems necessary for security against colluding adversaries) does not seem very amenable.

In light of the above, we choose to instead work with the delegation scheme of Chung, Kalai, and Vadhan [CKV10]. We briefly recall it below.

Delegation Scheme of [CKV10]. Let F be the functionality that we wish to outsource. The client picks a random r and computes $F(r)$ in the preprocessing phase. Next, in the online phase, after receiving the input x , the client picks a random bit b and sends either (x, r) or (r, x) to the server (depending on the bit b). The server must compute F on both x and r and return the responses back to the client. The client will check that $F(r)$ is correct and if so accept $F(x)$. Now, suppose x comes from the uniform distribution, then this protocol is a sound protocol and a cheating server can succeed only with probability $\frac{1}{2}$ (as he cannot distinguish (x, r) from (r, x) with probability better than $\frac{1}{2}$). For arbitrary distributions, this approach fails, but this can be rectified by having the client additionally pick a public key for an FHE scheme (in the preprocessing phase) and sending $(\text{Enc}_{pk}(x), \text{Enc}_{pk}(r))$ or $(\text{Enc}_{pk}(r), \text{Enc}_{pk}(x))$, depending on bit b in the online phase. The server will homomorphically evaluate the function F and respond back with $\text{Enc}_{pk}(F(x))$ and $\text{Enc}_{pk}(F(r))$. Now, this protocol is sound for arbitrary distributions of x . One can boost the soundness error to be negligibly small by picking random r_1, \dots, r_κ and having the client pick b_1, \dots, b_κ and send $(\text{Enc}_{pk}(x), \text{Enc}_{pk}(r_i))$ (or the other way around, depending on b_i). In order to make this protocol re-usable with the same values of r_1, \dots, r_κ , [CKV10] need to run this entire protocol under one more layer of fully homomorphic Encryption.

One might hope that we can apply this protocol directly by having the clients simulate the single client using multiparty computation. More specifically: the clients jointly generate the pre-processing information needed in the single-client case in the pre-processing phase of the protocol. In the online phase, the clients jointly generate the message sent by the single client (using a secure computation protocol) and similarly, in the offline phase,

the clients jointly run a secure computation to verify the results of the computation sent by the server. The main issue with this approach is that this solution requires the clients to interact (heavily) with each other in the online phase - which we believe, as discussed earlier, to be a significant drawback.

A Failed Approach. Towards that end, we consider the following approach. Let us consider the two-client setting first (where client D_i holds input x_i). As before the clients will jointly generate the information needed for pre-processing via a secure computation protocol. Our first idea is to have each client independently generate a bit b_i in the online phase and send either $(\text{Enc}_{pk}(x_i), \text{Enc}_{pk}(r_i))$ or $(\text{Enc}_{pk}(r_i), \text{Enc}_{pk}(x_i))$ to the server in the online phase. Now, the server instead of computing two outputs will compute 4 ciphertexts (since the server does not know bits b_1 and b_2 , it will compute ciphertexts corresponding to $F(x_1, x_2)$, $F(x_1, r_2)$, $F(r_1, x_2)$, and $F(r_1, r_2)$). The clients can then in the offline phase verify the appropriate responses of the server and accept $F(x_1, x_2)$ if all checks succeed.

Unfortunately, this solution *completely fails* in the case when a client (say D_2 colludes with the server). Very briefly, the main problem with the above approach is that it does not guarantee any *input independence*, a key requirement for a secure computation protocol. To see the concrete problem, recall that in order to realize the standard real/ideal paradigm for secure computation, we need to construct a simulator that simulates the view of the adversary in such a manner that output distributions of the real and ideal world executions are indistinguishable. The standard way such a simulator works is by “extracting” the input of the adversary in order to compute the “correct” protocol output (by querying the ideal functionality), and then “enforcing” this output in the protocol. The main issue that arises in the above approach is that we cannot guarantee that the adversary’s input extracted by the simulator is consistent with the output of the honest client in the real world execution. In particular, note that in the real world execution, the clients simply check whether the output of the server contains the correct $F(r_1, r_2)$ value, and if the check succeeds, then they decrypt the appropriate output value and accept it as their final output. Then, to argue indistinguishability of the real and ideal world executions, we would need to argue that the simulator extracts the specific input of the corrupted client that was used to compute the output by the (colluding) worker. However, the above approach provides *no* way of enforcing this requirement. As such, a colluding server and client pair can lead the simulator to compute an output which is inconsistent with the output generated by the server, in which case the simulator must abort. Yet, in the real world execution, the honest client would have accepted the output of the server. Thus, the simulator fails to generate an indistinguishable view of the adversary.

Indeed, the above attack demonstrates that when requiring simulation-based security against malicious coalitions, current techniques for single-client verifiable computation are not sufficient and new ideas are needed. The main contribution of our paper is a technique to overcome the above problem.

Our Solution. On careful inspection of the above approach, one can observe that it does provide some weak form of guarantee – that the server actually correctly computes the function F ; however, F is computed correctly w.r.t. “some” input for the corrupted client. In particular, the input of corrupted client may not be chosen independently of the honest client’s input.

In order to solve our problem, our main idea is to leverage the above weak guarantee by changing the functionality that is being delegated to the worker. Roughly speaking, we essentially “*delegate the delegation functionality*”. More concretely, we change the delegation functionality to $G(X, Y) = \text{Eval}_{pk}(X, Y, F), X, Y$, where X and Y are encryptions of the inputs x and y of the clients under the public key pk of the FHE scheme.

In order to understand why the above solution works, let us first consider an intermediate attempt where we delegate the functionality $\bar{F}(x, y) = F(x, y), x, y$. The underlying idea is to use the weak correctness guarantee (discussed above) to validate the input of the corrupted client. In more detail, note that if we delegate the functionality \bar{F} , then in the real world execution, we can have the clients perform the check (during the verification protocol) that the output value contains the same y value that is extracted by the simulator. Indeed, a priori, it may seem that this approach should indeed solve the above problem as we obtain a guarantee that the input of the malicious party in the output value (y) was indeed the same input used in the computation (as we are guar-

anteed correctness of the computation). Unfortunately, a subtle issue arises when trying to argue correctness of the simulator. Note that the above discussed check involves decrypting the output. While this is indeed possible in the real world execution, it is not clear how to perform the same check during simulation. Indeed, in the proof of security, we would need to rely on the semantic security of the underlying FHE scheme, which conflicts with performing decryption. We remark that the natural approach of having the simulator perform a related check (e.g., perform Eval operation of the FHE scheme to match the ciphertexts) rather than perform decryptions can be defeated by specific attacks by the adversary. We do not elaborate on this issue further here, but note that in order to argue correctness of simulation, we need to ensure that the verification checks performed in the honest execution and the simulated executions *must be the same*.

The above issue is resolved by delegating the functionality G (described above) instead of \bar{F} . The key idea is that instead of performing the aforementioned consistency check via decryption (of the single layer of FHE), we can now perform a similar check by first decrypting the outer layer, and then re-encrypting the input of the client (the clients are supposed to provide the encryption randomness in the verification protocol) and matching it with the values X and Y , respectively. The simulator can now be in possession of the outer layer FHE secret key since we can rely on the semantic security of the inner layer FHE.

The above description is somewhat oversimplified for lack of space. We refer the reader to the protocol description for more details.

Handling more than two clients. It is easy to see that the obvious extension of the above protocol to the case of n clients requires the server to compute and return 2^n values. Here we informally describe how to avoid this problem. Recall that each client picked a random bit b_i and sent either (x_i, r_i) or (r_i, x_i) , both doubly encrypted, depending on the bit b_i . To avoid the exponential blow up, we have the n clients jointly generate random bits b_1, \dots, b_n such that exactly one $b_i = 1$ (where i is random in $[n]$) and all other $b_j = 0, j \neq i$ (Doing this without interaction is a significant challenge, but we show that this can be achieved.). Now, the server only needs to compute $2n$ ciphertexts: n that encrypt $F(x_1, \dots, x_n)$ and n that encrypt $F(r_1, \dots, r_n)$. In this case, we can prove security of our protocol as long as at most a fraction of the clients are corrupted (even if they collude with the worker). More details in Section 5.1.

1.4 Related Work

The problem of efficiently verifying computations performed by an untrusted party has been extensively studied in the literature for the case of a *single* client outsourcing the computation to a server. Various approaches have been used: interactive solutions [GKR08], SNARG-based solutions [BCC88, Kil92, Mic94, BCCT12, GLR11, DFH12, Gro10, Lip12, GGPR12], and pre-processing model based solutions [GGP10, CKV10, AIK10]. The works of [KMR11] and [CKKC12] consider outsourcing of multi-party computation but consider only the case where adversarial parties do not collude with each other or a semi-honest setting. Finally, with a focus on minimizing interaction, the work of [CKKC13] considers non-interactive multi-client verifiable computation, but only consider soundness against a malicious server when clients are honest and privacy, independently, against a malicious server and a malicious client. For a more detailed description of related works, see Appendix A.

2 Preliminaries

2.1 Our Model

In this section, we present in detail, the computation and communication model as well as the security definition considered in this paper. For simplicity, we shall deal with the two-party case of our protocol first and then show how to extend this to the multi-party case. In other words, we consider the setting of 2 clients (or delegators) $\mathcal{D} = \{D_1, D_2\}$ who wish to jointly outsource the computation of any PPT function over their private inputs to a worker W . Specifically, we consider the case where the clients wish to perform arbitrarily many evaluations of a function F of their choice over different sets of inputs. Unlike the standard multiparty computation setting, we wish to ensure that the computation of each client is *independent* of the amount of computation needed

to compute F from scratch. The worker W , however, is expected to perform computation proportional to the size of the circuit representing F . Similar to standard multiparty computation, our corruption model allows an adversary to maliciously corrupt the worker and one of the clients (a subset of the clients in the multi-party case). Informally speaking, we require that no such adversary learns anymore than what can be learned from the inputs of the corrupted clients and their outputs (and any additional auxiliary information that the adversary may have).

Note that the above problem is non-trivial even in the setting where a single client wishes to outsource its computation to a worker. Specifically, all the known solutions require either the random oracle model, or appropriate non-black-box assumptions, or allow for a *one-time pre-processing phase* where the computation of the client(s) is allowed to be proportional to the size of the circuit representing F .¹ In this work, we wish to only rely on standard cryptographic assumptions; as such, we choose to work in the pre-processing model.

We now proceed to give a formal description of our model in the remainder of this section. We first present the syntax for a two-party verifiable computation protocol. We then define security as per the standard real/ideal paradigm for secure computation. Throughout this work, for simplicity of exposition, we assume that the function to be evaluated on the clients' inputs gives the same outputs to all clients. We note that the general scenario where the function outputs may be different for each client can be handled using standard techniques.

Two-party Verifiable Computation Protocol. A 2-party verifiable computation protocol between 2 clients (or delegators) $\mathcal{D} = \{D_1, D_2\}$ and a worker W consists of three phases, namely, (a) pre-processing phase, (b) online phase, and (c) offline phase. Here, the pre-processing phase is executed only once, while the online phase is executed every time the clients wish to compute F over a new set of inputs. The offline phase may also be executed each time following the online phase. Alternatively (and crucially), multiple executions of the offline phase can be *batched* together (i.e., the clients can outsource several (*not* fixed apriori) computations of F on different sets of inputs before they verify and obtain the results of these computations). We now describe the three phases in detail:

Preprocessing Phase: This is a *one-time* stage in which the clients compute some public, as well as private, information associated with the function F . The computation of each client in this phase is allowed to be proportional to the amount of computation needed to compute F from scratch. Clients are allowed to interact with each other in an arbitrary manner in this phase. Note that this phase is independent of the clients inputs and is executed only once.

Online Phase: In this phase, the clients interact with the server in a single round of communication. Let x_i the input held by client D_i . In order to outsource the computation of F (on a chosen set of inputs x_1, x_2) to the worker W , each client D_i *individually* prepares some public and private information about x_i and sends the public information to the worker. On receiving the encoded inputs from each client, W computes an encoded output and sends it to all the clients. Note that the clients do not directly interact with each other in this phase. This ensures that clients do not need to interact (and be available online) whenever they wish to perform some computation.

Offline Phase: On receiving the encoded output from the worker, the clients interact with each other to compute the actual output $F(x_1, x_2)$ and verify its correctness. We require that the computation of each client in this phase be independent of F . As we see, we will also minimize the interaction between the clients in this phase.

Note that the above protocol allows only for a single round of interaction between the clients and the worker. We require that the computation time of the clients in steps 2 and 3 above be, in total, less than the time required to compute F from scratch (in fact, in our protocol the computational complexity will be independent of F). Furthermore, we would like the worker to perform computation that is roughly the same as computing F .

¹In fact, this is the state of affairs even if we relax the security requirement to only output correctness, and do not require input privacy.

This completes the description of our computation and communication model. We now formally describe the key requirements from a 2-party verifiable computation protocol. Intuitively, we require that a verifiable computation protocol be both *efficient* and *secure*, in the sense as discussed below. We formally define both of these requirements below.

Security. To formally define security, we turn to the real/ideal paradigm for secure computation. We stress that we allow for an adversary that may corrupt either D_1 or D_2 (or none) *as well as* the worker. Since we consider the case of dishonest majority, we only obtain security with abort: i.e., the adversary first receives the function output, and then chooses whether the honest parties also learn the output, or to prematurely abort. Further, we only consider static adversaries, who choose the parties they wish to corrupt at the beginning of the protocol. Finally, we consider computational security, and thus we restrict our attention to PPT adversaries. We formally describe the ideal and real models of computation and then give our security definition in Appendix B.

Efficiency. Let the time required to compute function F be denoted by t_F ; we say that the *time complexity* of F is t_F .

Definition 1 We say that a verifiable computation protocol for computing a function F is efficient if it satisfies the following conditions:

- The running time of every client in the pre-processing phase is $\mathcal{O}(t_F)$.
- The running time of every client in the online phase is $o(t_F)$.
- The running time of the worker in the online phase is $\mathcal{O}(t_F)$.
- The running time of every client in the offline phase is $o(t_F)$.

2.2 Building Blocks

In our construction, we make use of several cryptographic primitives, listed as follows. We require pseudo-random functions, statistically binding commitments, fully homomorphic encryption, multi-key fully homomorphic encryption, the single client verifiable computation protocol from [CKV10] and a standard secure computation protocol. Due to lack of space, we describe these building blocks in detail in Appendix C.

3 Two-party Verifiable Computation Protocol

We now describe our two-party verifiable computation protocol Π for securely computing a functionality F . We proceed in two steps. First, in Section 3.1, we describe a one-time verifiable computation protocol where the pre-processing stage is useful only for one computation. Then, in Section 3.2, we show how our one-time construction can be modified to allow for multiple uses of the pre-processing phase.

3.1 One-Time Verifiable Computation Protocol

Let D_1, D_2 denote the two clients (or delegators) and W denote the worker.

Function outsourced to W . In order to securely compute function F over their private inputs, D_1 and D_2 outsource the computation of the following function \mathcal{G} to W :

Inputs: X_1, X_2 , where $X_i \leftarrow \text{Enc}_{pk}(x_i)$.

Output: $\mathcal{G}(X_1, X_2) = \text{Eval}_{pk}(X_1, X_2, F), X_1, X_2$.

We now proceed to describe the three phases of our protocol Π .

I. Pre-processing phase. In this phase, the clients interact with each other to perform the following computations:

1. D_1 and D_2 engage in the execution of a standard secure computation protocol Π_{fhe} to compute the (randomized) functionality F_{fhe} described as follows:
 - Generate key pairs $(sk, pk) \leftarrow \text{Gen}(1^\kappa)$ and $(SK, PK) \leftarrow \text{Gen}(1^\kappa)$ for the FHE scheme $(\text{Gen}, \text{Enc}, \text{Dec})$.
 - Compute 2-out-of-2 shares of the FHE secret keys sk, SK . That is, compute sk_1, sk_2 s.t. $sk_1 \oplus sk_2 = sk$, and SK_1, SK_2 s.t. $SK_1 \oplus SK_2 = SK$.
 - Output (pk, PK, sk_i, SK_i) to D_i .
2. D_1 and D_2 engage in the execution of a standard secure computation protocol Π_{prf} to compute the (randomized) functionality F_{prf} described as follows:
 - Sample keys K_1 and K_2 for a pseudo-random function $\text{PRF} : \{0, 1\}^\kappa \times \{0, 1\}^\kappa \rightarrow \{0, 1\}$.
 - For every $j \in [2]$, compute $(c_j^{\text{prf}}, d_j^{\text{prf}}) \leftarrow \text{COM}(K_i)$.
 - Output $(\{c_j^{\text{prf}}\}, \{d_i^{\text{prf}}\}, K_i)$ to D_i .
3. D_1 and D_2 engage in the execution of a standard secure computation protocol Π_{test} to compute the (randomized) functionality F_{test} described as follows. F_{test} takes as input the public key pk for FHE (as computed above) from D_1, D_2 and computes the following:
 - For every $i \in [2], j \in [n]$, generate random strings $r_{i,j}$ and compute $\widehat{R}_{i,j} \leftarrow \text{Enc}_{PK}(\text{Enc}_{pk}(r_{i,j}))$.
 - For every $j \in [n]$,
 - (a) Compute $\text{secret}_j = \text{Eval}_{PK}(\widehat{R}_{1,j}, \widehat{R}_{2,j}; \mathcal{G})$.
 - (b) Compute $(c_j^{\text{test}}, d_j^{\text{test}}) \leftarrow \text{COM}(\text{secret}_j)$.
 - (c) Choose random strings $d_{1,j}^{\text{test}}, d_{2,j}^{\text{test}}$ s.t. $d_j^{\text{test}} = d_{1,j}^{\text{test}} \oplus d_{2,j}^{\text{test}}$.
 - Output $(\widehat{R}_{i,j}, c_j^{\text{test}}, d_{i,j}^{\text{test}})$ to D_i .

(Note that the three steps above can be combined into a single secure computation protocol execution. We choose to split them into separate executions for simplicity of explanation and proof.)

II. Online phase. In this phase, the clients interact with the worker in a single round of communication to compute the functionality \mathcal{G} . For simplicity of exposition, we assume that the public keys (pk, PK) were given to W at the end of the pre-processing phase; we do not include them in the description below.

More specifically, this phase proceeds as follows:

$D_i \rightarrow W$: Let x_i denote the private input of D_i . The client D_i performs the following steps:

1. For every $j \in [n]$,
 - Compute $\widehat{X}_{i,j} \leftarrow \text{Enc}_{PK}(\text{Enc}_{pk}(x_i))$.
 - Let s be the session number. Then, compute bit $b_{i,j} \leftarrow \text{prf}_{K_i}(s||j)$. Let $(v_{i,j}^0, v_{i,j}^1)$ be such that $v_{i,j}^{b_{i,j}} = \widehat{X}_{i,j}$ and $v_{i,j}^{1-b_{i,j}} = \widehat{R}_{i,j}$.

D_i sends the tuple $\{v_{i,j}^0, v_{i,j}^1\}_{j=1}^n$ to W .

$W \rightarrow (D_1, D_2)$: On receiving the tuples $\{v_{i,j}^0, v_{i,j}^1\}_{j=1}^n$ from each client D_i , W performs the following steps. For every $j \in [n]$, homomorphically compute the following four values:

1. $z_j^0 = \text{Eval}_{PK}(v_{1,j}^0, v_{2,j}^0; \text{Eval}_{pk}(\cdot, \cdot, \mathcal{G}))$.
2. $z_j^1 = \text{Eval}_{PK}(v_{1,j}^0, v_{2,j}^1; \text{Eval}_{pk}(\cdot, \cdot, \mathcal{G}))$.
3. $z_j^2 = \text{Eval}_{PK}(v_{1,j}^1, v_{2,j}^0; \text{Eval}_{pk}(\cdot, \cdot, \mathcal{G}))$.
4. $z_j^3 = \text{Eval}_{PK}(v_{1,j}^1, v_{2,j}^1; \text{Eval}_{pk}(\cdot, \cdot, \mathcal{G}))$.

W sends the tuples $(v_{i,j}^0, v_{i,j}^1, z_j^\ell)$ to both the clients, where $\ell \in \{0, \dots, 3\}$, $i \in [2]$, $j \in [n]$.

III. Offline Phase. In this phase, the clients interact with each other to decode the output from the worker and verify its correctness. More specifically, D_1 and D_2 engage in an execution of a standard secure computation protocol Π_{ver} to compute the functionality F_{ver} described below.

Functionality F_{ver} . The input of client D_i to F_{ver} is the set of following values:

Public values: $s, pk, PK, c_{i,j}^{\text{test}}, c_i^{\text{prf}}, (v_{i,j}^0, v_{i,j}^1), z_j^\ell$, where $i \in [2], j \in [n], \ell \in \{0, \dots, 3\}$.

Private values: $x_i, \rho_{i,j}, sk_i, SK_i, d_{i,j}^{\text{test}}, d_i^{\text{prf}}$, where $j \in [n]$.

On receiving the above set of inputs from D_1 and D_2 , F_{ver} computes the following:

1. Match all the public input values of D_1 and D_2 . If any of the corresponding values are not equal, output \perp .
2. For every $i \in [2]$, if $\perp \leftarrow \text{OPEN}(c_i^{\text{prf}}, d_i^{\text{prf}})$, then output \perp . Otherwise, let $K_i \leftarrow \text{OPEN}(c_i^{\text{prf}}, d_i^{\text{prf}})$.
3. For every $j \in [n]$, do the following:
 - (a) If $\perp \leftarrow \text{OPEN}(c_j^{\text{test}}, d_{1,j}^{\text{test}} \oplus d_{2,j}^{\text{test}})$, then output \perp . Otherwise, let $\text{secret}_j^* = \text{OPEN}(c_j^{\text{test}}, d_{1,j}^{\text{test}} \oplus d_{2,j}^{\text{test}})$.
 - (b) For every $i \in [2]$, compute $b_{i,j} = \text{PRF}_{K_i}(s||j)$. Let $p_j = 2b_{1,j} + b_{2,j} + 1$. If $\text{secret}_j^* \neq z_j^{4-p_j}$, then output \perp .
 - (c) Compute $(Y_j[0], Y_j[1], Y_j[2]) \leftarrow \text{Dec}_{SK_1 \oplus SK_2}(z_j^{p_j})$. For any $i \in [2]$, do the following:
 - If $Y_j[i] \neq \text{Dec}_{SK_1 \oplus SK_2}(v_{i,j}^{b_{i,j}})$, then output \perp .
 - If $Y_j[i] \neq \text{Enc}_{pk}(x_i; \rho_{i,j})$, then output \perp .
4. Output $y = \text{Dec}_{sk_1 \oplus sk_2}(Y_1[0])$.

This completes the description of our protocol. We now claim the following:

Theorem 1 *Assuming the existence of a fully homomorphic encryption scheme, protocol Π is a secure and efficient one-time verifiable computation protocol for any efficiently computable functionality f .*

It follows from observation that protocol Π is an efficient verifiable computation protocol as per Definition 1. In Section 4, we prove its security as per Definition 2.

3.2 Many-time Verifiable Computation

We now explain how our one-time verifiable computation protocol Π described in previous subsection can be extended to allow for multiple uses of the pre-processing phase.

Similar to [GGP10, CKV10], we achieve reusability of the pre-processing phase by executing the online phase under an additional layer of FHE (and performing necessary decryptions in the offline phase). However, since we wish to avoid interaction between the clients during the online phase, a priori, it is not clear how the new public key must be chosen during each execution of the online phase. We resolve this issue by making use of a multi-key homomorphic encryption scheme (MFHE) (see Section C.3 for definition). More specifically, we make the following changes to our one-time verifiable computation protocol:

Online Phase:

1. Each client D_i generates a key pair $(MPK_i, MSK_i) \leftarrow \text{MGen}(1^\kappa)$ for the MFHE scheme. For every $j \in [n]$, it computes $\widehat{X}_{i,j} \leftarrow \text{MEnc}_{MPK_i}(\widehat{X}_{i,j})$ and $\widehat{R}_{i,j} \leftarrow \text{MEnc}_{MPK_i}(\widehat{R}_{i,j})$, where $\widehat{X}_{i,j}$ and $\widehat{R}_{i,j}$ are as defined earlier. The values sent to W are $(\widehat{v}_{i,j}^0$ and $\widehat{v}_{i,j}^1)$, where $\widehat{v}_{i,j}^{b_{i,j}} = \widehat{X}_{i,j}$ and $\widehat{v}_{i,j}^{1-b_{i,j}} = \widehat{R}_{i,j}$.
2. The worker W now computes the following values: for every $j \in [n]$,
 - (a) $\widehat{z}_j^0 = \text{MEval}_{MPK_1, MPK_2}(\widehat{v}_{1,j}^0, \widehat{v}_{2,j}^0, \text{Eval}_{PK}(\cdot, \cdot, \text{Eval}_{pk}(\cdot, \cdot, \mathcal{G})))$.
 - (b) $\widehat{z}_j^1 = \text{MEval}_{MPK_1, MPK_2}(\widehat{v}_{1,j}^0, \widehat{v}_{2,j}^1, \text{Eval}_{PK}(\cdot, \cdot, \text{Eval}_{pk}(\cdot, \cdot, \mathcal{G})))$.
 - (c) $\widehat{z}_j^2 = \text{MEval}_{MPK_1, MPK_2}(\widehat{v}_{1,j}^1, \widehat{v}_{2,j}^0, \text{Eval}_{PK}(\cdot, \cdot, \text{Eval}_{pk}(\cdot, \cdot, \mathcal{G})))$.
 - (d) $\widehat{z}_j^3 = \text{MEval}_{MPK_1, MPK_2}(\widehat{v}_{1,j}^1, \widehat{v}_{2,j}^1, \text{Eval}_{PK}(\cdot, \cdot, \text{Eval}_{pk}(\cdot, \cdot, \mathcal{G})))$.

Offline Phase:

1. Each client D_i uses (MPK_1, MPK_2) and MSK_i as additional inputs in the protocol Π_{ver} .
2. For every $j \in [n]$, the functionality F_{ver} computes the values $z_j^{p_j} \leftarrow \text{MDec}_{MSK_1, MSK_2}(\widehat{z}_j^{p_j})$ and $z_j^{4-p_j} \leftarrow \text{MDec}_{MSK_1, MSK_2}(\widehat{z}_j^{4-p_j})$, and then performs the same computations as described earlier.

This completes the description of the many-time verifiable computation protocol. We discuss its security in Section D.2.

4 Proof of Security

In order to prove Theorem 1, we will first construct a PPT simulator \mathcal{S} that simulates the view of any adversary \mathcal{A} who corrupts one of the clients and the worker. We will then argue that the output distributions in the real and ideal world executions are computationally indistinguishable. We start by describing the construction of \mathcal{S} in Section 4.1. We complete the proof by arguing the correctness of simulation in Appendix D.1.

4.1 Description of Simulator

Without loss of generality, below we assume that the client D_2 and the worker W are corrupted by the adversary. We will denote them as D_2^* and W^* , respectively. The opposite case where D_1 and W are corrupted can be handled analogously.

We describe how the simulator \mathcal{S} works in each of the three phases:

Pre-processing phase:

1. Let S_{fhe} denote the simulator for the two-party computation protocol Π_{fhe} . In the first step, \mathcal{S} runs S_{fhe} with D_2^* to generate a simulated execution of Π_{fhe} . During the simulation, when S_{fhe} makes a query to the ideal functionality F_{fhe} , \mathcal{S} evaluates F_{fhe} on its own using fresh randomness and returns the output to S_{fhe} . Let pk, PK denote the output public keys in this protocol.
2. Let S_{prf} denote the simulator for the two-party computation protocol Π_{prf} . In the second step, \mathcal{S} runs S_{prf} with D_2^* to generate a simulated execution of Π_{prf} . During the simulation, when S_{prf} makes a query to the ideal functionality F_{prf} , \mathcal{S} evaluates F_{prf} on its own using fresh randomness and returns the output to S_{prf} . Let $(\{c_j^{\text{prf}}\}, d_i^{\text{prf}}, K_i)$ denote the output of D_i , where each variable is defined in the same way as in the protocol description in the previous section.
3. Let S_{test} denote the simulator for the two-party computation protocol Π_{test} . In the final step of the pre-processing phase, \mathcal{S} runs S_{test} with D_2^* to generate a simulated execution of Π_{test} . During the simulation, when S_{test} makes a query to the ideal functionality F_{test} , \mathcal{S} evaluates F_{test} on its own using fresh randomness and returns the output to S_{test} . Let $(\widehat{R}_{i,j}, c_j^{\text{test}}, d_{i,j}^{\text{test}})$ denote the output of D_i , where each variable is defined in the same way as in the protocol description in the previous section.

Online Phase. In this phase, \mathcal{S} works in essentially the same manner as the honest client D_1 , except that it uses the all zeros string 0^κ as its input and uses random bits $b_{i,j}$ (instead of pseudorandom bits).

\mathcal{S} behaves honestly for the rest of the online phase. Let $(v_{i,j}^0, v_{i,j}^1, z_j^\ell)$ be the tuple \mathcal{S} receives from W^* , where $\ell \in \{0, \dots, 3\}$, $i \in [2]$, $j \in [n]$.

Offline Phase. Let S_{ver} denote the simulator for the two-party computation protocol Π_{ver} . In the offline phase, \mathcal{S} runs the simulator S_{ver} with the corrupted client D_2^* to generate a simulated execution of Π_{ver} . At some point during the simulation, S_{ver} makes a query to the ideal functionality F_{ver} with some input (say) \tilde{Z} , \mathcal{S} does the following:

1. Parse \tilde{Z} as follows:

Public values: $\tilde{s}, \tilde{pk}, \tilde{PK}, \tilde{c}_{i,j}^{\text{test}}, \tilde{c}_i^{\text{prf}}, (\tilde{v}_{i,j}^0, \tilde{v}_{i,j}^1), \tilde{z}_j^\ell$, where $i \in [2], j \in [n], \ell \in \{0, \dots, 3\}$.

Private values: $\tilde{x}_2, \tilde{\rho}_{2,j}, \tilde{b}_{2,j}, \tilde{s}k_2, \tilde{SK}_2, \tilde{d}_{2,j}^{\text{test}}, \tilde{d}_i^{\text{prf}}$, where $j \in [n]$.

2. Perform the verification checks in the same manner as F_{ver} , except the check regarding the PRF key K_1 . (In particular, use the bits $b_{1,j}$ directly computed for the checks. If any of the checks fail, then output \perp to S_{ver} . Else, if all of the checks succeed, then query the ideal functionality for F on input \tilde{x}_2 . Let y be the output. Return y as the output to S_{ver} .

On receiving the output value y from \mathcal{S} , S_{ver} continues the simulation of Π_{ver} where it forces the output y on \mathcal{A} , and finally stops. At this point, \mathcal{S} stops as well and outputs the entire simulated view of \mathcal{A} .

5 Extensions

5.1 Handling Multiple Clients

In this section, we shall show how we can extend our two-party verifiable computation protocol to obtain an n -party verifiable computation that is secure against a constant fraction (α) of corrupted clients (who can collude arbitrarily with a corrupted server).

Let us first recall a basic idea that we used in our protocol for two-party verifiable computation. The two clients D_1 and D_2 picked random strings r_1 and r_2 in the pre-processing phase and computed an encryption of $\mathcal{G}(r_1, r_2)$. In the online phase, D_1 picked a random bit b_1 and sent either (x_1, r_1) or (r_1, x_1) , both doubly encrypted, depending on the bit b_1 . D_2 picked a random b_2 and sent either (x_2, r_2) or (r_2, x_2) , both doubly encrypted, depending on the bit b_2 . Let the ciphertexts sent by D_1 be denoted by (v_1^0, v_1^1) and those sent by D_2 be denoted by (v_2^0, v_2^1) respectively. The server W , then responded with 4 ciphertexts, computed homomorphically: namely, $z^0 = \text{MEval}_{PK_1, PK_2}(v_1^0, v_2^0; \text{Eval}_{pk}(\cdot, \cdot, \mathcal{G}))$, $z^1 = \text{MEval}_{PK_1, PK_2}(v_1^0, v_2^1; \text{Eval}_{pk}(\cdot, \cdot, \mathcal{G}))$, $z^2 = \text{MEval}_{PK_1, PK_2}(v_1^1, v_2^0; \text{Eval}_{pk}(\cdot, \cdot, \mathcal{G}))$, and $z^3 = \text{MEval}_{PK_1, PK_2}(v_1^1, v_2^1; \text{Eval}_{pk}(\cdot, \cdot, \mathcal{G}))$ (In the real protocol, the clients actually picked κ such random strings each and the server sent back 4κ ciphertexts evaluated homomorphically back to the clients; but for simplicity, we will only consider one such instance here.). Note that out of the 4 ciphertexts, one of them contained an encryption of $\mathcal{G}(r_1, r_2)$, that was used for verification, and one of them contained an encryption of $\mathcal{G}(x_1, x_2)$, that was used to obtain the result of the computation.

Now, suppose we tried to extend the above idea as it is, to the n -party case. That is, the n clients each pick r_1, \dots, r_n in the pre-processing phase and compute $\mathcal{G}(r_1, \dots, r_n)$. In the online phase, each client D_i picks a private random bit b_i and sends either (x_i, r_i) or (r_i, x_i) , both doubly encrypted, depending on the bit b_i . Unfortunately now, since the server does not (and cannot) know the bits b_i , the server must send back 2^n ciphertexts in order to be sure that he has sent back both an encryption of $\mathcal{G}(r_1, \dots, r_n)$ (to be used for verification by the n clients) as well as encryption of $\mathcal{G}(x_1, \dots, x_n)$ (to be used to obtain the result of the computation by the n clients). This solution ends up having a non-polynomial time complexity for all parties.

Our idea to solve the above problem is to have the n clients jointly generate random bits b_1, \dots, b_n such that exactly 1 $b_i = 1$ (where i is random in $[n]$) and all other $b_j = 0, j \neq i$. Now, the server only needs to compute

$2n$ ciphertexts in order to be sure that he has returned the required ciphertexts. In other words, W computes the following $2n$ sets of ciphertexts:

1. The n ciphertexts, one of which encrypts $\mathcal{G}(x_1, \dots, x_n)$, namely:

- (a) $z^0 = \text{MEval}_{PK_1, PK_2}(v_1^1, v_2^0, \dots, v_n^0; \text{Eval}_{pk}(\cdot, \cdot, \dots, \cdot, \mathcal{G}))$,
- (b) $z^1 = \text{MEval}_{PK_1, PK_2}(v_1^0, v_2^1, v_3^0, \dots, v_n^0; \text{Eval}_{pk}(\cdot, \cdot, \cdot, \dots, \cdot, \mathcal{G}))$,
- ...
- (c) $z^n = \text{MEval}_{PK_1, PK_2}(v_1^0, \dots, v_{n-1}^0, v_n^1; \text{Eval}_{pk}(\cdot, \dots, \cdot, \cdot, \mathcal{G}))$

2. and the n ciphertexts, one of which encrypts $\mathcal{G}(r_1, \dots, r_n)$, namely:

- (a) $z^{n+1} = \text{MEval}_{PK_1, PK_2}(v_1^0, v_2^1, \dots, v_n^1; \text{Eval}_{pk}(\cdot, \cdot, \dots, \cdot, \mathcal{G}))$,
- (b) $z^{n+2} = \text{MEval}_{PK_1, PK_2}(v_1^1, v_2^0, v_3^1, \dots, v_n^1; \text{Eval}_{pk}(\cdot, \cdot, \cdot, \dots, \cdot, \mathcal{G}))$,
- ...
- (c) $z^{2n} = \text{MEval}_{PK_1, PK_2}(v_1^1, \dots, v_{n-1}^1, v_n^0; \text{Eval}_{pk}(\cdot, \dots, \cdot, \cdot, \mathcal{G}))$

The above idea ensures that the complexity of the worker remains polynomial (the complexity of the clients are still independent of F except for the pre-processing phase). Two (linked) issues remain to be addressed: 1) How do the clients generate the bits b_1, \dots, b_n with the required distribution without interacting with each other? and 2) the security of the protocol. These issues are addressed in Appendix E.

5.2 Minimizing Interaction in Offline Phase

Recall that in the offline phase, the clients need to execute a secure computation protocol in order to verify and obtain the output of the computation. Note that since we work in the pre-processing model, we can use a specific multi-party computation protocol in order to reduce the round complexity of the clients in this phase. More specifically, we can use any secure computation protocol, even one that makes use of pre-processing, so long as this pre-processing is re-usable for multiple runs of the protocol. Such a protocol exists due to the construction of Asharov *et al.* [AJLA⁺12], which is a 2-round secure computation protocol in the re-usable pre-processing model with CRS (note that in our case, the clients can compute the CRS needed for this protocol during the initial pre-processing phase). Using this protocol, we can obtain a multi-party verifiable computation protocol in which the round complexity of the clients in the offline phase is 2.

References

- [AF07] Masayuki Abe and Serge Fehr. Perfect NIZK with adaptive soundness. In *TCC*, pages 118–136, 2007.
- [AIK10] B. Applebaum, Y. Ishai, and E. Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. In *ICALP*, 2010.
- [AJLA⁺12] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold fhe. In *EUROCRYPT*, pages 483–501, 2012.
- [BCC88] Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *J. Comput. Syst. Sci.*, 37(2):156–189, 1988.
- [BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *ITCS*, pages 326–349, 2012.

- [BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. Fully homomorphic encryption without bootstrapping. In *ITCS*, 2012.
- [BOGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC*, pages 1–10, 1988.
- [BV11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. In *FOCS*, pages 97–106, 2011.
- [CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *STOC*, pages 11–19, 1988.
- [CKKC12] Seung Geol Choi, Jonathan Katz, Ranjit Kumaresan, and Carlos Cid. Multi-user non-interactive verifiable computation. *ACITA Conference*, 2012.
- [CKKC13] Seung Geol Choi, Jonathan Katz, Ranjit Kumaresan, and Carlos Cid. Multi-client non-interactive verifiable computation. In *TCC*, pages 499–518, 2013.
- [CKV10] Kai-Min Chung, Yael Tauman Kalai, and Salil P. Vadhan. Improved delegation of computation using fully homomorphic encryption. In *CRYPTO*, pages 483–501, 2010.
- [DFH12] Ivan Damgård, Sebastian Faust, and Carmit Hazay. Secure two-party computation with low communication. In *TCC*, pages 54–74, 2012.
- [Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009.
- [GGP10] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *CRYPTO*, pages 465–482, 2010.
- [GGPR12] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct nizks without pcps. *IACR Cryptology ePrint Archive*, 2012:215, 2012.
- [GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In *STOC*, pages 113–122, 2008.
- [GLR11] Shafi Goldwasser, Huijia Lin, and Aviad Rubinfeld. Delegation of computation without rejection problem from designated verifier CS-proofs. *IACR Cryptology ePrint Archive*, 2011:456, 2011.
- [GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC*, pages 218–229, 1987.
- [Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In *ASIACRYPT*, pages 321–340, 2010.
- [GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *STOC*, pages 99–108, 2011.
- [HPS98] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. Ntru: A ring-based public key cryptosystem. In *ANTS*, pages 267–288, 1998.
- [Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *STOC*, pages 723–732, 1992.

- [KMR11] Seny Kamara, Payman Mohassel, and Mariana Raykova. Outsourcing multi-party computation. *IACR Cryptology ePrint Archive*, 2011:272, 2011.
- [KMR12] Seny Kamara, Payman Mohassel, and Ben Riva. Salus: a system for server-aided secure function evaluation. In *ACM Conference on Computer and Communications Security*, pages 797–808, 2012.
- [Lip12] Helger Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In *TCC*, volume 7194 of *Lecture Notes in Computer Science*, pages 169–189. Springer, 2012.
- [LTV12] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *STOC*, pages 1219–1234, 2012.
- [Mic94] Silvio Micali. Cs proofs (extended abstracts). In *FOCS*, pages 436–453, 1994.
- [Nao89] Moni Naor. Bit commitment using pseudo-randomness. In *CRYPTO*, pages 128–136, 1989.
- [Nao03] Moni Naor. On cryptographic assumptions and challenges. In *CRYPTO*, pages 96–109, 2003.
- [PRV12] Bryan Parno, Mariana Raykova, and Vinod Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In *TCC*, pages 422–439, 2012.
- [Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC*, pages 84–93, 2005.
- [Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *FOCS*, pages 160–164, 1982.

A Related works

Interactive Solutions. Goldwasser et al. [GKR08] show how to build an interactive proof between a client and a server to verify arbitrary polynomial time computations in almost linear time. Because of the interactive nature, this protocol is not suited to the multi-client case (as we discussed above this would require the clients to be all present and interacting with the server during the computation – our model enforces a single message exchanged between server and client during the online phase)².

SNARGs. A class of solutions is based on *succinct non-interactive arguments* (or SNARGs): (computationally sound [BCC88]) proofs that are very short and very efficient to verify, regardless of the complexity of the function being evaluated. Solutions of this type are usually constructed using *Probabilistically Checkable Proofs* (PCPs), long proofs that the verifier can check in only very few places (in particular only a constant number of bits of the proofs are needed for NP languages). Kilian [Kil92] showed how to use PCPs to construct interactive succinct arguments by committing to the entire PCP string using a Merkle tree. Micali [Mic94] removed the interaction by use of a random oracle. Recent work [BCCT12, GLR11, DFH12] has replaced the random oracle with an “extractable collision-resistant hash functions” (ECRHs), a *non-falsifiable* [Nao03], assumption that any algorithm that computes an image of the ECRH must “know” the corresponding pre-image.

There are alternative constructions of SNARGs based on different forms of arithmetization of Boolean computations used together with cryptographic constructions based on bilinear maps (e.g. [Gro10, Lip12, GGPR12]). Those protocols also rely on non-falsifiable “knowledge” assumptions over the cryptographic groups used in the

²A non-interactive argument for a restricted class of functions is also presented in [GKR08]. We did not investigate if this could be turned into a multi-client non-interactive protocol (though we suspect it could, when coupled with an FHE), because the focus of this paper is a general solution for arbitrary polynomial computations.

constructions. We note that following [AF07, GW11] such “knowledge” assumptions seem unavoidable when using SNARGs.

As we pointed out in the previous Section, SNARGs coupled with FHE would yield a conceptually simple protocol for the multi-client case, but at the cost of using either the random oracle or a falsifiable assumption. Our protocol instead relies on standard cryptographic assumptions (in particular the existence of FHE).

Verifiable Computation. Gennaro, Gentry and Parno [GGP10] and subsequent works [CKV10, AIK10] present a *verifiable computation* (VC) scheme that allows a client to outsource any efficiently computable function to a worker and verify the worker’s computation in constant time, where the worker’s complexity is only *linear* in the size of the circuit, assuming a one-time expensive preprocessing phase depending on the function being outsourced. We use their approach, and most specifically we use the protocol from [CKV10] as our starting point. [PRV12] show how to construct a protocol for outsourcing computation of a smaller class of functions starting from any attribute-based encryption scheme. Their solution does not handle input privacy however.

Server-Aided Secure Function Evaluation. The first work to explicitly consider outsourced computation in the multi-client case is by Kamara et al [KMR11] (see also [KMR12] which reports on some optimizations of [KMR11] and implementation results). The main limitation of these works is a *non-colluding* adversarial model where the server and the clients may maliciously depart from the protocol specifications but without a common strategy or communication. A simpler protocol is also presented in [CKKC12] but it assumes only semi-honest parties. We stress that our work is the first to achieve full simulation security in the most stringent adversarial model.

B Security definition

We now formally describe the ideal and real models of computation and then give our security definition.

IDEAL WORLD. In the ideal world, there is a trusted party T that computes the desired functionality F on the inputs of all parties. Unlike the standard ideal world model, here we allow for multiple evaluations of the functionality on different sets of inputs. An execution of the ideal world consists of (unbounded) polynomial number of repetitions of the following:

Inputs: D_1 and D_2 have inputs x_1 and x_2 respectively. The worker has no input. All parties send their inputs to the trusted party T . Additionally, corrupted parties may change their inputs before sending them to T .

Trusted party computes output: T computes $F(x_1, x_2)$.

Adversary learns output: T returns $F(x_1, x_2)$ to \mathcal{A} .

Honest parties learn output: \mathcal{A} prepares a list of (possibly empty) set of honest clients that should get the output and sends it to T . T sends $F(x_1, x_2)$ to this set of honest clients and \perp to other honest clients in the system.

Outputs: All honest parties output whatever T gives them. Corrupted parties, wlog, output \perp . The *view* of \mathcal{A} in the ideal world execution above includes the inputs of corrupt parties, the outputs of all parties, as well as the entire view of all corrupt parties in the system. \mathcal{A} can output any arbitrary function of its view and we denote the random variable consisting of this output, along with the outputs of all honest parties, by $\text{IDEAL}_{F, \mathcal{A}}(x_1, x_2)$.

REAL WORLD. In the real world, there is no trusted party and the parties interact directly with each other according to a protocol Π . Honest parties follow all instructions of Π , while adversarial parties are coordinated by a single adversary \mathcal{A} and may behave arbitrarily. At the conclusion of the protocol, honest clients compute their output as prescribed by the protocol.

For any set of adversarial parties (that may include a corrupt worker and a corrupt D_1 or D_2) controlled by \mathcal{A} and protocol Π for computing function F , we let $\text{REAL}_{\pi, \mathcal{A}}(x_1, x_2)$ be the random variable denoting the output of \mathcal{A} in the real world execution above, along with the output of the honest parties. $\text{REAL}_{\pi, \mathcal{A}}(x_1, x_2)$ can be an arbitrary function of the view of \mathcal{A} that consists of the inputs (and random tape) of corrupt parties, the outputs of all parties in the protocol, as well as the entire view of all corrupt parties in the system.

Security Definition. Intuitively, we require that for every adversary in the real world, there exists an adversary in the ideal world, such that the views of these two adversaries are computationally indistinguishable. Formally,

Definition 2 *Let F and Π be as above. We say that Π is a secure verifiable computation protocol for computing F if for every PPT adversary \mathcal{A} that corrupts either D_1 or D_2 , and additionally possibly corrupts the worker, in the real model, there exists a PPT adversary \mathcal{S} (that corrupts the same set of parties as \mathcal{A}) in the ideal execution, such that:*

$$\text{IDEAL}_{F, \mathcal{A}}(x_1, x_2) \stackrel{c}{=} \text{REAL}_{\pi, \mathcal{A}}(x_1, x_2)$$

C Building Blocks

We now describe the building blocks we require in our protocol.

C.1 Statistically Binding Commitments

We shall make use of statistically binding commitments in our protocol. A statistically binding commitment consists of two probabilistic algorithms: **COM** and **OPEN**. **COM** takes as input a message m from the sender and outputs a “commitment” to m , denoted by c to the receiver and a “decommitment” to m , denoted by d , to the sender. **OPEN** takes as input c and d and outputs a message m (denoting the message that was committed to) or outputs \perp denoting reject. The correctness property of a commitment scheme requires that for all honestly executed **COM** and **OPEN**, we have that $\text{OPEN}(\text{COM}(m)) = m$, except with negligible probability. Informally, a statistically binding commitment scheme has the security property that no (computationally unbounded) sender can commit to a message m and have it decommit to some other message. In other words, no computationally unbounded sender can come up with a commitment c and two decommitments d and d' such that $\text{OPEN}(c, d) = m$ and $\text{OPEN}(c, d') = m'$ for different m and m' . In our protocol, we shall make use of Naor’s two-round statistically binding commitment scheme [Nao89]. At a high level, the commitment scheme, based on any pseudorandom generator, G , from κ bits to 3κ bits, works as follows: in the commitment phase, the receiver sends a random 3κ bit string, r to the sender. The sender picks a seed s (of length κ) to the pseudorandom generator at random and sends $G(s)$ to commit to 0 and $G(s) \oplus r$ to commit to 1. The decommitment is simply the bit and the seed s . This scheme is statistically binding and computationally hiding.

C.2 Fully homomorphic Encryption

In our constructions, we shall make use of a fully homomorphic encryption (FHE) scheme [Gen09, BV11, BGV12]. An FHE scheme consists of four algorithms: (a) a key generation algorithm $\text{Gen}(1^\kappa)$ that takes as input the security parameter and outputs a public key/secret key pair (pk, sk) , (b) a randomized encryption algorithm $\text{Enc}_{pk}(m)$ that takes as input the public key and a message m and produces ciphertext c , (c) a decryption algorithm $\text{Dec}_{sk}(c)$ that takes as input the secret key, ciphertext c and produces a message m , and (d) a *deterministic*³ evaluation algorithm $\text{Eval}_{pk}(c, F)$ that takes as input a ciphertext c (that encrypts a message m), the public key, and (the circuit description of) a PPT function F and produces a ciphertext c^* .

The correctness of the encryption, decryption, and evaluation algorithms require that for all key pairs output by Gen , $\text{Dec}_{sk}(\text{Enc}_{pk}(m)) = m$, for all m (except with negligible probability) and that $\text{Dec}_{sk}(\text{Eval}_{pk}(\text{Enc}_{pk}(m), F)) = F(m)$, for all m and PPT F , (except with negligible probability). The compactness property of an FHE scheme requires the following: let $c^* \leftarrow \text{Eval}_{pk}(c, F)$. There exists a polynomial

³The Eval algorithm need not be deterministic in general, but we require that the algorithm be deterministic. There are plenty of such schemes available based on a variety of assumptions.

P such that $|c^*| \leq P(\kappa)$. (In other words, the size of c^* is independent of the size of circuit description of F .) Finally, the security definition for an FHE scheme is that of standard semantic security for encryption schemes.

We note that FHE schemes are known to exist from a variety of cryptographic assumptions such as the learning with errors (LWE) assumption [Reg05].

C.3 Multikey Fully Homomorphic Encryption

In our construction, we will use a multikey fully homomorphic encryption (MFHE) scheme [LTV12] to avoid interaction between the players during the online phase of our protocol. An MFHE scheme is defined as a four-tuple of algorithms (MGen, MEnc, MEval, MDec): (a) a key generation algorithm $(MPK, MSK) \leftarrow \text{MGen}(1^\kappa)$ that takes as input the security parameter and outputs a public key/secret key pair (MPK, MSK) , (b) a randomized encryption algorithm $c \leftarrow \text{MEnc}_{MPK}(m)$ that takes as input the public key and a message m and produces a ciphertext c , (c) a decryption algorithm $m \leftarrow \text{MDec}_{MSK_1, \dots, MSK_n}(c)$ that takes as input n secret keys MSK_i and a ciphertext c , and outputs a message m , and (d) a *deterministic* evaluation algorithm $c^* \leftarrow \text{MEval}_{MPK_1, \dots, MPK_n}(c_1, \dots, c_n, C)$ that takes as a input (the circuit description of) a PPT function F , a list of ciphertexts c_1, \dots, c_n along with the corresponding public keys MPK_1, \dots, MPK_n , and produces a new ciphertext c^* .

An MFHE scheme must satisfy the following two requirements: (a) **Correctness**: For every $c^* \leftarrow \text{MEval}_{MPK_1, \dots, MPK_n}(c_1, \dots, c_n, F)$, where $c_i \leftarrow \text{MEnc}_{MPK_i}(m_i)$, it must be that $\text{MDec}_{MSK_1, \dots, MSK_n}(c^*) = C(m_1, \dots, m_t)$. (b) **Compactness**: Let $c^* \leftarrow \text{MEval}_{MPK_1, \dots, MPK_n}(c_1, \dots, c_n, F)$. Then, there exists a polynomial P such that $|c^*| \leq P(\kappa, n)$.

The security definition for an MFHE scheme is that of standard semantic security for encryption schemes. We remark that in the above description, for simplicity of notation, we do not explicitly mention “evaluation keys”, and simply assume that they are part of the public keys.

We note that an MFHE scheme was recently constructed by López-Alt et al. [LTV12] based on NTRU [HPS98].

C.4 Single-Client Verifiable Computation

As a building block for our solution we use the recently proposed method for single-client verifiable computations [CKV10]. For concreteness, we briefly describe the solution in [CKV10] which can be based on any Fully Homomorphic Encryption (FHE) scheme [Gen09]. The high level idea for their protocol to outsource a function F is as follows. The client picks a random r and computes $F(r)$ in the preprocessing phase. Next, in the online phase, after receiving the input x , the client picks a random bit b and sends either (x, r) or (r, x) to the server (depending on the bit b). The server must compute F on both x and r and return the responses back to the client. The client will check that $F(r)$ matches the pre-computed value and if so accept the other response as the correct $F(x)$. Now, suppose x comes from the uniform distribution, then this protocol is a sound protocol and a cheating server can succeed only with probability $\frac{1}{2}$ (as he cannot distinguish (x, r) from (r, x) with probability better than $\frac{1}{2}$). For arbitrary distributions, this approach fails, but this can be rectified by having the client additionally pick a public key for an FHE scheme (in the preprocessing phase) and sending $(\text{Enc}_{pk}(x), \text{Enc}_{pk}(r))$ or $(\text{Enc}_{pk}(r), \text{Enc}_{pk}(x))$, depending on bit b in the online phase. The server will homomorphically evaluate the function F and respond back with $\text{Enc}_{pk}(F(x))$ and $\text{Enc}_{pk}(F(r))$. Now, this protocol is sound for arbitrary distributions of x (as a cheating server cannot distinguish $(\text{Enc}_{pk}(x), \text{Enc}_{pk}(r))$ from $(\text{Enc}_{pk}(r), \text{Enc}_{pk}(x))$). One can boost the soundness error to be negligibly small by picking random r_1, \dots, r_κ and having the client pick b_1, \dots, b_κ and send $(\text{Enc}_{pk}(x), \text{Enc}_{pk}(r_i))$ (or the other way around, depending on b_i). The client will check that all values of $F(r_i)$ were correct and that the κ different values for $F(x)$ were identical and if so, accept $F(x)$. In order to make this protocol re-usable with the same values of r_1, \dots, r_κ , [CKV10] need to run this entire protocol under one more layer of fully homomorphic Encryption.

C.5 Secure Computation

We make use of a two-party secure computation protocol (between parties D_1 and D_2). We make use of such a protocol that is secure in the standard ideal/real world paradigm.

IDEAL WORLD. In the ideal world, there is a trusted party T that computes the desired functionality F on the inputs of the two parties. An execution of the ideal world consists of the following:

Inputs: D_1 and D_2 have inputs x_1 and x_2 respectively and send their inputs to the trusted party T . Additionally, a corrupted party may change its input before sending them to T .

Trusted party computes output: T computes $F(x_1, x_2)$.

Adversary learns output: T returns $F(x_1, x_2)$ to \mathcal{A} (here, either D_1 or D_2 is controlled by the adversary \mathcal{A}).

Honest parties learn output: \mathcal{A} determines if the honest party should get the output and sends this to T . T sends $F(x_1, x_2)$ to this honest party (if the adversary says so) and \perp otherwise.

Outputs: Honest parties output whatever T gives them. Corrupted parties, wlog, output \perp . The *view* of \mathcal{A} in the ideal world execution above includes the inputs of corrupt parties, the outputs of all parties, as well as the entire view of all corrupt parties in the system. \mathcal{A} can output any arbitrary function of its view and we denote the random variable consisting of this output, along with the outputs of all honest parties, by $\text{IDEAL}_{F, \mathcal{A}}(x_1, x_2)$.

REAL WORLD. In the real world, there is no trusted party and the parties interact directly with each other according to a protocol Π_{2pc} . Honest parties follow all instructions of Π_{2pc} , while adversarial parties are coordinated by a single adversary \mathcal{A} and may behave arbitrarily. At the conclusion of the protocol, honest clients compute their output as prescribed by the protocol.

For any set of adversarial parties (that is, corrupt D_1 or D_2) controlled by \mathcal{A} and protocol Π_{2pc} for computing function F , we let $\text{REAL}_{\pi, \mathcal{A}}(x_1, x_2)$ be the random variable denoting the output of \mathcal{A} in the real world execution above, along with the output of the honest parties. $\text{REAL}_{\pi, \mathcal{A}}(x_1, x_2)$ can be an arbitrary function of the view of \mathcal{A} that consists of the inputs (and random tape) of corrupt parties, the outputs of all parties in the protocol, as well as the entire view of all corrupt parties in the system.

SECURITY DEFINITION. Intuitively, we require that for every adversary in the real world, there exists an adversary in the ideal world, such that the views of these two adversaries are computationally indistinguishable. Formally,

Definition 3 *Let F and Π_{2pc} be as above. Protocol Π_{2pc} is a secure protocol for computing F if for every PPT adversary \mathcal{A} that corrupts either D_1 or D_2 , in the real model, there exists a PPT adversary \mathcal{S}_{2pc} (that corrupts the same party as \mathcal{A}) in the ideal execution, such that:*

$$\text{IDEAL}_{F, \mathcal{A}}(x_1, x_2) \stackrel{c}{\equiv} \text{REAL}_{\pi, \mathcal{A}}(x_1, x_2)$$

D Proof details

D.1 Indistinguishability of the Views

In order to prove Theorem 1, we consider a series of hybrid experiments $\mathcal{H}_0, \dots, \mathcal{H}_4$, where \mathcal{H}_0 represents the real world execution, while \mathcal{H}_4 corresponds to the simulated execution in the ideal world. We will show that each consecutive pair of hybrid experiments are computationally indistinguishable. We can therefore conclude that \mathcal{H}_0 and \mathcal{H}_4 are computationally indistinguishable, as required.

Experiment \mathcal{H}_0 . This experiment corresponds to the real world execution. The simulator simply uses the honest party input and runs the honest party algorithm in the protocol execution.

Experiment \mathcal{H}_1 . This experiment is the same as \mathcal{H}_0 , except that in the pre-processing phase, \mathcal{S} runs the simulators S_{fhe} , S_{prf} and S_{test} instead of running the honest party algorithm. Note that the functionalities F_{fhe} , F_{prf} and F_{test} are still computed honestly, in the same manner as in \mathcal{H}_0 .

Indistinguishability of \mathcal{H}_0 and \mathcal{H}_1 : From the security of the two-party computation protocols Π_{fhe} , Π_{prf} and Π_{test} , it immediately follows that the output distributions of \mathcal{H}_0 and \mathcal{H}_1 are computationally indistinguishable.

Experiment \mathcal{H}_2 . This experiment is the same as \mathcal{H}_1 , except that in the offline-phase, \mathcal{S} runs the simulator S_{ver} instead of running the honest party algorithm. \mathcal{S} answers the output query of S_{ver} by computing F_{ver} in the same manner as description of \mathcal{S} .

Indistinguishability of \mathcal{H}_1 and \mathcal{H}_2 : From the security of the two-party computation protocol Π_{ver} , it immediately follows that the output distributions of \mathcal{H}_1 and \mathcal{H}_2 are computationally indistinguishable.

Experiment \mathcal{H}_3 . This experiment is the same manner as \mathcal{H}_2 except that \mathcal{S} computes the bits $b_{i,j}$ for the honest party P_i as random bits (instead of computing them pseudorandomly).

Indistinguishability of \mathcal{H}_2 and \mathcal{H}_3 : Follows immediately from the security of PRF.

Experiment \mathcal{H}_3 . This experiment is the same as \mathcal{H}_2 , except that now, in order to compute the final output of F_{ver} , \mathcal{S} queries the ideal functionality F instead of performing decryption in the final step.

Indistinguishability of \mathcal{H}_2 and \mathcal{H}_3 : We now claim that hybrids \mathcal{H}_2 and \mathcal{H}_3 are statistically indistinguishable. Towards contradiction, suppose that there exists a distinguisher that can distinguish between the output distributions of \mathcal{H}_2 and \mathcal{H}_3 with inverse polynomial probability $p(\kappa)$. Now, note that the only difference between \mathcal{H}_2 and \mathcal{H}_3 is the manner in which the final outputs are computed. In other words, the existence of such a distinguisher implies that the outputs computed in \mathcal{H}_3 and \mathcal{H}_4 are different. However, note that conditioned on the event that worker W performs the computation correctly, then the checks performed by F_{ver} corresponding to the inputs of the parties (i.e., step no 2(c) in the description of F_{ver}) guarantee that the outputs in both experiments must be the same. Thus, from the check 2(b) of F_{ver} , we now have that the existence of such a distinguisher D implies that with inverse polynomial probability $p'(\kappa)$, the worker W is able to provide incorrect answers at positions p_j , and correct answers at positions $4 - p_j$, for all $j \in [n]$. We now obtain a contradiction using the soundness lemma of Chung et al. [CKV10].

In more detail, we now consider an experiment G where the simulator interacts with the server as in \mathcal{H}_4 , and then stops the experiment at the end of the online phase. That is, in \mathcal{H}_3 , for every $j \in [n]$, \mathcal{S} prepares each $\widehat{X_{i,j}} \leftarrow \text{Enc}_{PK}(\text{Enc}_{pk}(x_i))$ and $\widehat{R_{i,j}} \leftarrow \text{Enc}_{PK}(R_i)$. Now, consider an alternate experiment G' that is the same as G , except that \mathcal{S} now prepares $\widehat{X_{i,j}} \leftarrow \text{Enc}_{PK}(R_i)$. Then, the following equation follows from the semantic security of the (outer layer) FHE scheme:

$$\begin{aligned} & \Pr[W \text{ correct on } (\widehat{R_{i,1}}, \dots, \widehat{R_{i,n}}) \text{ and incorrect on } (\widehat{X_{i,1}}, \dots, \widehat{X_{i,j}}) \text{ in } G] \\ & \leq \Pr[W \text{ correct on } (\widehat{R_{i,1}}, \dots, \widehat{R_{i,n}}) \text{ and incorrect on } (\widehat{X_{i,1}}, \dots, \widehat{X_{i,j}}) \text{ in } G'] + \text{negl}(\kappa) \end{aligned}$$

Note that to obtain the above equation, we rely on the fact that the function outsourced is a PPT function, and thus we can check whether W is correct or incorrect by executing the Eval algorithm. Note that the simulator knows the positions where the Eval checks must be performed since it knows the PRF key of the adversary and can therefore compute its random bits $b_{i^*,j}$.

Now, it is easy to see that:

$$\Pr[W \text{ correct on } (\widehat{R_{i,1}}, \dots, \widehat{R_{i,n}}) \text{ and incorrect on } (\widehat{X_{i,1}}, \dots, \widehat{X_{i,j}}) \text{ in } G'] \leq \frac{1}{2^n}$$

Thus, combining the above two equations, we arrive at a contradiction. We refer the reader to [CKV10] for more details.

Experiment \mathcal{H}_4 . This experiment is the same as \mathcal{H}_3 , except that instead of encrypting the input of P_1 honestly, \mathcal{S} computes $\widehat{X}_{1,1}, \dots, \widehat{X}_{1,n}$ as encryptions of the all zeros string. Note that this experiment corresponds to the ideal world.

Indistinguishability of \mathcal{H}_3 and \mathcal{H}_4 : From the semantic security of the (inner layer) FHE scheme (Gen, Enc, Dec, Eval), it immediately follows that the output distributions of \mathcal{H}_3 and \mathcal{H}_4 are computationally indistinguishable. In more detail, assume for contradiction that there exists a PPT distinguisher D that can distinguish with non-negligible probability between the output distributions of \mathcal{H}_3 and \mathcal{H}_4 . Then, we construct an adversary B that breaks the semantic security for the FHE scheme. Adversary B takes a public key pk from the challenger C of the FHE scheme and then runs the simulator \mathcal{S} to generate an output distribution for D . Specifically, B follows the same strategy as \mathcal{S} , except that in the pre-processing phase, \mathcal{S} forces pk as the inner layer public key. B then sends vectors \vec{m}_0, \vec{m}_1 as its challenge messages to C , where each element in \vec{m}_0 is set to x_1 , and each element in \vec{m}_1 is set to the all zeros string. On receiving the challenge ciphertext vector from C , adversary B simply uses them to continue the rest of the simulation as \mathcal{S} does. Finally, B outputs whatever D outputs. Now, note that if the challenge ciphertexts correspond to \vec{m}_0 , then the resultant output distribution is same as in experiment \mathcal{H}_3 , otherwise, it is the same as in \mathcal{H}_4 . Thus, by definition D (and therefore B) must succeed in distinguishing with non-negligible probability. Thus, we arrive at a contradiction.

D.2 Security of the Many-time Verifiable Computation Scheme

So far, we have shown that our protocol is one-time secure (namely, that soundness holds when one execution of the online and offline phases are executed). We now proceed to show that the protocol is many-time secure (i.e., we can run (unbounded) polynomially many online and offline phases after one run of the pre-processing phase). As in the works of [GGP10, CKV10], we work in a model in which if the result of some computation returned by the server is rejected by any of the clients, the clients execute a new pre-processing phase and pick new parameters. To prove our security, let us first recall how [CKV10] go from one-time to many-time security. [CKV10] show that if we have a one-time delegation scheme, then this can be converted into a many-time delegation scheme simply by executing the entire protocol under another layer of fully encryption. A fresh public key for the FHE scheme is chosen for every execution by the client in the online phase. Note that the way we achieve multi-time security is also similar - the clients independently pick a fresh public key for the multi-key homomorphic encryption scheme of [LTV12] and execute the one-time protocol under this layer of fully homomorphic encryption. The proof that our protocol is also multi-time secure is quite similar to that of [CKV10]; however, there are a few subtle changes that we need to make.

To see these changes, let us first understand the idea behind the proof of [CKV10]. [CKV10] reduce the security of the multi-time scheme to that of the one-time scheme. That is, given an adversary that breaks the security of the multi-time scheme, they construct an adversary that breaks the security of the one-time scheme. Both adversaries execute the pre-processing phase in exactly the same manner. Let L be an upper bound on the total number of times the one-time stage is executed by the adversary. The one-time scheme adversary that they construct picks one of these executions, say i^{th} , at random and chooses to break the one-time security of that execution. In all other executions, the adversary will “simulate” the protocol by sending encryptions of all-zeroes, instead of sending the encryption of the actual message that is a function of \widehat{X} (which is an encryption of the client’s input), \widehat{R} (which is an encryption of the client’s secret state used to verify the protocol), and the secret bit b (which is chosen fresh in every execution). In these executions (all executions other than the i^{th}), one can show (via the semantic security of the FHE scheme) that the messages sent in the real and simulated executions are indistinguishable. An important point here is that the client never rejects any of these executions here and always proceeds with the computation as if it accepted it. In the i^{th} execution, the adversary will pick the public key and secret key for the FHE scheme on its own. Now, the adversary encrypts the query under this public key and once it obtains the response of the worker from the many-time adversary, it decrypts it using the secret key to obtain the response that the adversarial worker in the one-time game must produce.

We shall also follow the same overall strategy. The one-time adversary that we build will execute the pre-

processing phase in exactly the same manner as the many-time adversary. We will also pick one of the L executions at random and choose to break the one-time security of that execution. However, unlike [CKV10], we cannot simulate the other executions by sending encryptions of all-zeroes. This is because, if we did so, then the verification performed by the clients in the offline phase will necessarily fail and in the event that one of the clients colludes with the server (which is unique to our setting), this information will be learned by the server and we will not be able to continue with simulation. The way around it is to simulate other executions by sending the encryption of the message exactly as we would do in a real run of the protocol, that is as a function of \hat{X} , \hat{R} , and the secret bit b , except that we shall replace the r encrypted in \hat{R} , and encrypt all-zeroes instead (note that b is not part of the secret state as this varies from execution to execution). Note that by doing this we do not use the secret state that is carried between executions anywhere in our simulation. Now, in the offline phase, our adversary will run the simulator for the two-party computation protocol and force the clients to output “accept” and the result of the computation to be $F(x_1, x_2)$. Now, note that client never rejects any of these executions and always proceeds with the computation as if it accepted it. The rest of the simulation works exactly the same as in the case of [CKV10] and with these slight changes, our proof of security goes through. We now present more details.

Let \mathcal{B} be an adversary (controlling a cheating D_2^* and W^*) that succeeds with non-negligible probability in breaking the security when executing the online phase multiple times. We shall construct an adversary \mathcal{A} that also succeeds in breaking the security with non-negligible probability, but when executing the online phase only once.

- \mathcal{A} executes the pre-processing phase in exactly the same manner as \mathcal{B} . In other words, \mathcal{A} executes the pre-processing phase exactly as the simulator described in Section 4.1 does.
- Next, let L be an upper bound on the total number of online executions run by adversary \mathcal{B} . \mathcal{A} picks an index $1 \leq i \leq L$ at random, and this is the execution that it will use to distinguish between the real and ideal worlds.
 - In every execution $k \neq i$, \mathcal{A} does as follows: \mathcal{A} uses the honest client D_1 's input in that execution, say x_1 , in computing the messages sent by D_1 in the online phase. However, instead of using the random value $\widehat{R}_{1,j}$ in the online phase (for $1 \leq j \leq n$), \mathcal{A} will use encryptions of the all-zero string instead. \mathcal{A} will execute the rest of the online phase exactly as an honest D_1 would (that is, \mathcal{A} picks random bits $b_{1,j}$, for $1 \leq j \leq n$ and sends the ordered encryption tuples (of x_1 and 0) to W).
 - In the i^{th} execution, \mathcal{A} does as follows: \mathcal{A} picks the keys for the outermost layer of the FHE scheme on its own (both the public and secret keys (pk^*, sk^*)). \mathcal{A} upon receiving a query from the client D_1 in the one-time execution, will prepare the query using this value and use the public key pk^* to encrypt the query. \mathcal{A} will send this value to \mathcal{B} . Upon receiving the response from \mathcal{B} , \mathcal{A} will decrypt it using sk^* and send this value as the value sent by the malicious worker controlled by \mathcal{A} in the one-time execution.

Note that if \mathcal{B} terminates the game before the i^{th} execution, then \mathcal{A} aborts and gives up.

- In the offline phase, for all executions $k \neq i$, \mathcal{A} will execute the simulator, S_{ver} , for the two-party computation protocol, Π_{ver} along with the corrupted client \mathcal{B} to generate a simulated execution of Π_{ver} . At some point during the simulation, S_{ver} will make a query to the ideal functionality F_{ver} with some input (say) \tilde{Z} . At this point \mathcal{A} will simply return $F(x_1, x_2)$ as the result of the computation to S_{ver} . On receiving this output value y from \mathcal{A} , S_{ver} continues the simulation of Π_{ver} where it forces the output y on \mathcal{B} . In the offline phase for the i^{th} execution, \mathcal{A} will execute the protocol honestly in this phase.

The proof that \mathcal{A} also succeeds with non-negligible probability, when \mathcal{B} succeeds with non-negligible probability follows via a standard hybrid argument. At a high level, note that we can argue about the success probability

of \mathcal{A} only in the case when \mathcal{A} guesses correctly the *first* execution when \mathcal{B} will execute the protocol maliciously. This is because, if \mathcal{B} executes the protocol maliciously for some execution $q < i$, then since \mathcal{A} will simulate the output of the computation to be $F(x_1, x_2)$ (in the offline phase), when in the real world the output of the computation maybe different causing \mathcal{B} to distinguish between the real and ideal worlds. Hence, let us consider the case when \mathcal{A} guesses correctly the first execution when \mathcal{B} is malicious. Note that this happens with probability $\frac{1}{L}$. Now, given that the first instance that \mathcal{B} is malicious is only in the i^{th} execution, we have that \mathcal{B} is honest in the first $i - 1$ executions. In these executions, we can show indistinguishability of the different hybrids (where we replace a real execution with a simulated execution in a step-by-step manner) via a simple hybrid argument as the only difference between the real and ideal executions is that we are replacing encryptions of $\widehat{R_{1,j}}$ in the online phase (for $1 \leq j \leq n$) with encryptions of all zeroes and we are simulating the offline phase so that it outputs the value $F(x_1, x_2)$ to the adversary. The first change is indistinguishable due to the semantic security of the FHE scheme, while the second change is indistinguishable since the adversary is indeed honest in this execution and an honest execution indeed does evaluate to $F(x_1, x_{\text{@}})$ (from the indistinguishability of the simulated two-party computation protocol from the real protocol with same output value, this indistinguishability follows). Hence, if \mathcal{B} succeeds with probability $p_{\mathcal{B}}$, then \mathcal{A} succeeds with probability at least $\frac{p_{\mathcal{B}}}{L} + \text{negl}$. We leave further details to the full version of the paper.

E Multi-party verifiable computation

Let us have the clients pick the bits b_i at random from a distribution that outputs 1 with probability $\frac{1}{n}$ and 0 otherwise (such a distribution can easily be sampled; simply pick $\log n$ bits uniformly at random and outputting 1 iff all $\log n$ bits are 1). Let us look at the completeness of the protocol in this case. When all parties are honest, the probability that exactly one $b_i = 1$ and all other b_i 's are 0 is $n \times \frac{1}{n} \times (1 - \frac{1}{n})^{n-1}$ which is $\geq \frac{1}{e}$. The probability that there is a completeness error is bounded by $1 - \frac{1}{e}$. So, if the clients repeat the above protocol (in parallel) κ number of times, then the probability that *none* of the repetitions succeed will be negligibly small in κ . The clients can check only the run of the protocol that succeeded during the offline verification phase, and obtain the result of the computation.

The problem with this approach is that a set of corrupted clients can claim that their random bits are such that $b_i = 1$ for a corrupted client. Now, with constant probability, b_j will be 0 for all honest clients. This means that the colluding adversarial clients along with the server will have complete knowledge of the bits b_1, \dots, b_n in this case and hence soundness will be completely defeated.

We get around this problem as follows. We repeat the protocol (in parallel), a total of $2en^2\kappa$ number of times. But now, a client D_i will accept the output of the computation, iff there are at least κ number of repetitions in which $b_i = 1$ and $b_j = 0$ for all $j \neq i$ (this will be checked in the secure computation protocol run during the offline phase). Let us first analyze the completeness of this protocol. Note that, for a particular b_i , the probability that $b_i = 1$ and all other b_j 's are 0 is at least $\frac{1}{en}$. Hence, if run $2en\kappa$ executions in parallel, except with negligible probability (in κ , via the Chernoff bound), we get that there will be κ number of repetitions in which $b_i = 1$ and $b_j = 0$ for all $j \neq i$. Since we are running $2en^2\kappa$ parallel repetitions, except with negligible probability, for all clients D_i , this condition will be met.

Now, let us analyze the security of this protocol. Note that the adversarial set of clients (totally αn clients for constant $0 < \alpha < 1$) cannot simply set their bits such that one of their b bits is always 1 in all parallel repetitions of the protocol. Hence, the adversarial clients must set their bits to 0 in at least $(1 - \alpha)\kappa n$ executions. Now, note that in these repetitions of the protocol, the adversary has no idea as to which honest clients bit $b_i = 1$ (this can be shown using the same techniques as in the proof of the two-party protocol). Since the adversarial client can force a wrong output (by colluding with the corrupted worker) only by guessing this, the probability with which this happens is $(\frac{1}{(1-\alpha)n})^{(1-\alpha)n\kappa}$, which is negligible in the security parameter κ . Hence, a set of adversarial clients succeed in forcing a wrong output only with negligible probability.

An Equational Approach to Secure Multi-party Computation*

Daniele Micciancio[†]

Stefano Tessaro[‡]

December 4, 2012

Abstract

We present a novel framework for the description and analysis of secure computation protocols that is at the same time mathematically rigorous and notationally lightweight and concise. The distinguishing feature of the framework is that it allows to specify (and analyze) protocols in a manner that is largely independent of time, greatly simplifying the study of cryptographic protocols. At the notational level, protocols are described by systems of mathematical equations (over domains), and can be studied through simple algebraic manipulations like substitutions and variable elimination. We exemplify our framework by analyzing in detail two classic protocols: a protocol for secure broadcast, and a verifiable secret sharing protocol, the second of which illustrates the ability of our framework to deal with probabilistic systems, still in a purely equational way.

1 Introduction

Secure multiparty computation (MPC) is a cornerstone of theoretical cryptography, and a problem that is attracting increasingly more attention in practice too due to the pervasive use of distributed applications over the Internet and the growing popularity of computation outsourcing. The area has a long history, dating back to the seminal work of Yao [29] in the early 1980s, and a steady flow of papers contributing extensions and improvements that lasts to the present day (starting with the seminal works [12, 6, 3] introducing general protocols, and followed by literally hundreds of papers). But it is fair to say that MPC has yet to deliver its full load of potential benefits both to the applied and theoretical cryptography research communities. In fact, large portions of the research community still see MPC as a highly specialized research area, where only the top experts can read and fully understand the highly technical research papers routinely published in mainstream crypto conferences. Two main obstacles have kept, so far, MPC from becoming a more widespread tool to be used both in theoretical and applied cryptography: the prohibitive computational cost of executing many MPC protocols, and the inherent complexity of the models used to describe the protocols themselves. Much progress has been made in improving the efficiency of the first protocols [29, 12, 6] in a variety of models and with respect to several complexity measures, even

*This material is based on research sponsored by DARPA under agreement number FA8750-11-C-0096. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the U.S. Government.

[†]University of California, San Diego. daniele@cs.ucsd.edu.

[‡]Massachusetts Institute of Technology. tessaro@csail.mit.edu

leading to concrete implementations (cf. e.g. [19, 4, 17, 7, 24, 11]). However, the underlying models to describe and analyze security properties are still rather complex.

What makes MPC harder to model than traditional cryptographic primitives like encryption, is the inherently distributed nature of the security task being addressed: there are several distinct and mutually distrustful parties trying to perform a joint computation, in such a way that even if some parties deviate from the protocol, the protocol still executes in a robust and secure way.

The difficulty of properly modeling secure distributed computation is well recognized within the cryptographic community, and documented by several definitional papers attempting to improve the current state of the art [22, 9, 10, 23, 2, 18, 15, 20]. Unfortunately, the current state of the art is still pretty sore, with definitional/modeling papers easily reaching encyclopedic page counts, setting a very high barrier of entry for most cryptographers to contribute or actively follow the developments in MPC research. Moreover, most MPC papers are written in a semi-formal style reflecting an uncomfortable trade-off between the desire of giving to the subject the rigorous treatment it deserves and the implicit acknowledgment that this is just not feasible using the currently available formalisms (and even more so, within the page constraints of a typical conference or even journal publication.) At the other end, recent attempts to introduce abstractions [20] are too high level to deliver a precise formal language for protocol specification. The goal of this paper is to drastically change this state of affairs, by putting forward a model for the study of MPC protocols that is both concise, rigorous, and still firmly rooted in the intuitive ideas that pervade most past work on secure computation and most cryptographers know and love.

1.1 The simulation paradigm

Let us recall the well known and established simulation paradigm that underlies essentially all MPC security definitions. Cryptographic protocols are typically described by several component programs P_1, \dots, P_n executed by n participating parties, interconnected by a communication network N , and usually rendered by a diagram similar to the one in Figure 1 (left): each party receives some input x_i from an external environment, and sends/receives messages s_i, r_i from the network. Based on the external inputs x_i , and the messages s_i, r_i transmitted over the network N , each party produces some output value y_i which is returned to the outside world as the visible output of running the protocol. The computational task that the protocol is trying to implement is described by a single monolithic program F , called the “ideal functionality”, which has the same input/output interface as the system consisting of P_1, \dots, P_n and N , as shown in Figure 1 (right). Conceptually, F is executed by a centralized entity that interacts with the individual parties through their local input/output interfaces x_i/y_i , and processes the data in a prescribed and trustworthy manner. A protocol P_1, \dots, P_n correctly implements functionality F in the communication model provided by N if the two systems depicted in Figure 1 (left and right) exhibit the same input/output behavior.

Of course, this is not enough for the protocol to be secure. In a cryptographic context, some parties can get corrupted, in which case an adversary (modeled as part of the external execution environment) gains direct access to the parties’ communication channels s_i, r_i and is not bound to follow the instructions of the protocol programs P_i . Figure 2 (left) shows an execution where P_3 and P_4 are corrupted. The simulation paradigm postulates that whatever can be achieved by a concrete adversary attacking the protocol, can also be achieved by an idealized adversary S (called the simulator) attacking the ideal functionality F . In Figure 2 (right), the simulator takes over the role of P_3 and P_4 , communicating for them with the ideal functionality, and recreating the attack of a real adversary by emulating an interface that exposes the network communication channels of P_3

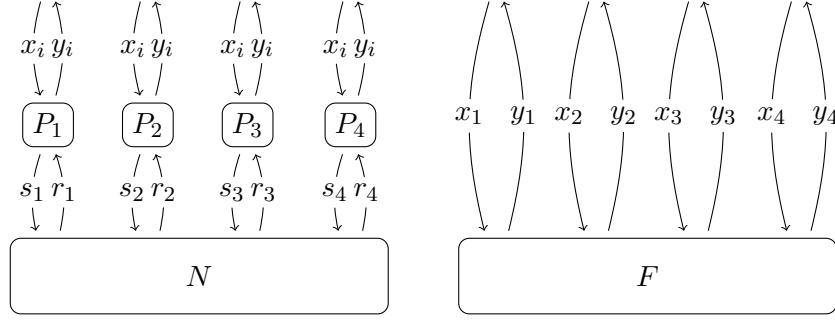


Figure 1: A multiparty protocol P_1, \dots, P_4 with communication network N (left) implementing a functionality F (right).

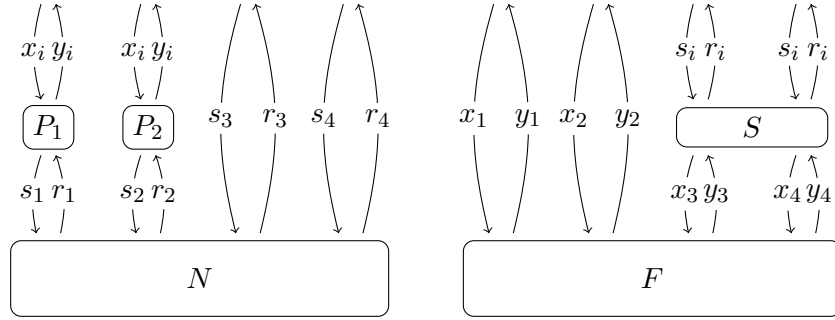


Figure 2: Simulation based security. The protocol P_1, \dots, P_4 is a secure implementation of functionality F if the system (left) exposed to an adversary that corrupts a subset of parties (say P_3 and P_4) is indistinguishable from the one system (right) recreated by a simulator S interacting with F .

and P_4 . The protocol P_1, \dots, P_n securely implements functionality F if the systems described on the left and right of Figure 2 are functionally equivalent: no adversary (environment) connecting to the external channels $x_1, y_1, x_2, y_2, s_3, r_3, s_4, r_4$ can (efficiently) determine if it is interacting with the system described in Figure 2 (left) or the one in Figure 2 (right). In other words, anything that can be achieved corrupting a set of parties in a real protocol execution, can also be emulated by corrupting the same set of parties in an idealized execution where the protocol functionality F is executed by a trusted party in a perfectly secure manner.

This is a very powerful idea, inspired by the seminal work on zero knowledge proof systems [13], and embodied in many subsequent papers about MPC. But, of course, as evocative the diagrams in Figure 2 may be, they fall short of providing a formal definition of security. In fact, a similar picture can be drawn to describe essentially any of the secure multiparty computation models proposed so far at a very abstract level, but the real work is in the definition of what the blocks and communication links connecting them actually represent. Traditionally, building on classical work from computational complexity on interactive proof systems, MPC is formalized

by modeling each block by an interactive Turing machine (ITM), a venerable model of sequential computation extended with some “communication tapes” used to model the channels connecting the various blocks. Unfortunately, this only provides an adequate model for the local computation performed by each component block, leaving out the most interesting features that distinguish MPC from simpler cryptographic tasks: computation is distributed and the concurrent execution of all ITMs needs to be carefully orchestrated. In a synchronous communication environment, where local computations proceed in lockstep through a sequence of rounds, and messages are exchanged only between rounds, this is relatively easy. But in asynchronous communication environments like the Internet, dealing with concurrency is a much trickier business. The standard approach to deal with concurrency in asynchronous distributed systems is to use nondeterminism: a system does not describe a single behavior, but a set of possible behaviors corresponding to all possible interleavings and message delivery orders. But nondeterminism is largely incompatible with cryptography, as it allows to break any cryptographic function by nondeterministically guessing the value of a secret key. As a result, cryptographic models of concurrent execution resort to an adversarially and adaptively chosen, but deterministic, message delivery order: whenever a message is scheduled for transmission between two component, it is simply queued and an external scheduling unit (which is also modeled as part of the environment) is notified about the event. While providing a technically sound escape route from the dangers of mixing nondeterministic concurrency with cryptography, this approach has several shortcomings:

- Adding a scheduler further increases the complexity of the system, making simulation based proofs of security even more technical.
- It results in a system that in many respects seems overspecified: as the goal is to design a robust system that exhibits the prescribed behavior in any execution environment, it would seem more desirable to abstract the scheduling away, rather than specifying it in every single detail of a fully sequential ordering of events.
- Finally, the intuitive and appealing idea conveyed by the diagrams in Figure 2 is in a sense lost, as the system is now more accurately described by a collection of isolated components all connected exclusively to the external environment that orchestrates their executions by scheduling the messages.

1.2 Our work

In this paper we describe a model of distributed computation that retains the simplicity and intuitiveness conveyed by the diagrams in Figures 1 and 2, and still it is both mathematically rigorous and concise. In other words, we seek a model where the components P_i, N, F, S occurring in the description and analysis of a protocol, and the systems obtained interconnecting them, can be given a simple and precise mathematical meaning. The operation of composing systems together should also be well defined, and satisfy a number of useful and intuitive properties, e.g., the result of connecting several blocks together does not depend on the order in which the connections are made. (Just as we expect the meaning of a diagram to be independent of the order in which the diagram was drawn.) Finally, it should provide a solid foundation for equational reasoning, in the sense that equivalent systems can be replaced by equivalent systems in any context.

Within such a framework, the proof that protocols can be composed together should be as simple as the following informal argument. (In fact, given the model formally defined in the rest of the

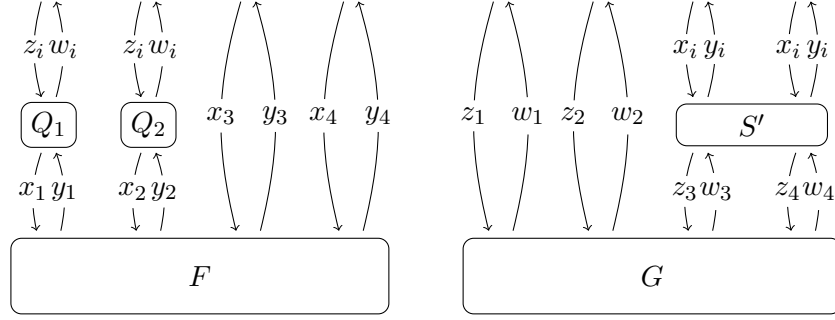


Figure 3: A protocol Q_i implementing G in the F -hybrid model.

paper, the following is actually a rigorous proof that our definition satisfies a universal composability property.) Say we have a protocol P_1, \dots, P_n securely implementing ideal functionality F using a communication network N , and also a protocol Q_1, \dots, Q_n in the F -hybrid model (i.e., an idealized model where parties can interact through functionality F) that securely implements functionality G . The security of the second protocol is illustrated in Figure 3.

Then, the protocol obtained simply by connecting P_i and Q_i together is a secure implementation of G , in the standard communication model N . Moreover, the simulator showing that the composed protocol is secure is easily obtained simply by composing the simulators for the two component protocols. In other words, we want to show that an adversary attacking the real system described in Figure 4 (left) is equivalent to the composition of the simulators attacking the ideal functionality G as described in Figure 4 (right).

This is easily shown by transforming Figure 4 (left) to Figure 4 (right) in two steps, going through the hybrid system described in Figure 5. Specifically, first we use the security of P_i to replace the system described in Figure 2 (left) with the one in Figure 2 (right). This turns the system in Figure 4 (left) into the equivalent one in Figure 5. Next we use the security of Q_i to substitute the system in Figure 3 (left) with the one in Figure 3 (right). This turns Figure 5 into Figure 4 (right).

While the framework proposed in this paper allows to work with complex distributed systems with the same simplicity as the informal reasoning described in this section, it is quite powerful and flexible. For example, it allows to model not only protocols that are universally composable, but also protocols that retain their security only when used in restricted contexts. For simplicity, in this paper we focus on *perfectly secure* protocols against *unbounded* adversaries, as this already allows us to describe interesting protocols that illustrate the most important feature of our framework: the ability to design and analyze protocols without explicitly resorting to the notion of time and sequential scheduling of messages. Moreover, within the framework of universal composability, it is quite common to design perfectly secure protocols in a hybrid model that offers idealized versions of the cryptographic primitives, and then resorting to computationally secure cryptographic primitives only to realize the hybrid model. So, a good model for the analysis of perfect or statistical security can already be a useful and usable aid for the design of more general computationally secure protocols. Natively extending our framework to statistically or computationally secure protocols is also an attractive possibility. We consider the perfect/statistical/computational security dimension

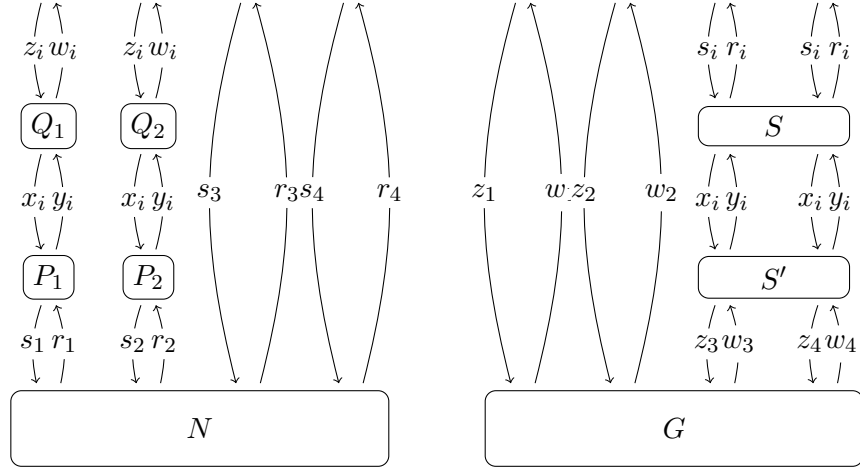


Figure 4: Protocol composition. Security is proved using a hybrid argument.

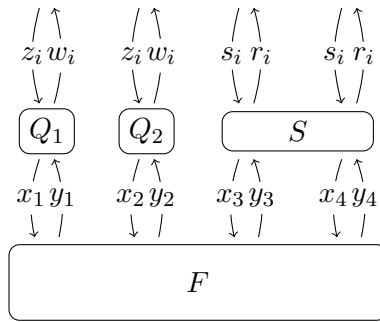


Figure 5: Hybrid system to prove the security of the composed protocol

as being mostly orthogonal to the issues dealt with in this paper, and we believe the model described here offers a solid basis for extensions in that direction.

1.3 Techniques

In order to realize our vision, we introduce a computational model in which security proofs can be carried out without explicitly dealing with the notion of time. Formally, we associate to each communication channel connecting two components the set of all possible “channel histories”, partially ordered according to their information content or temporal ordering. The simplest example is the set of all finite sequences M^* of messages from some underlying message space, ordered according to the prefix ordering relation. The components of the system are then modeled as functions mapping input histories to output histories. The functions are subject to some natural conditions, e.g., monotonicity: receiving more input values can only results in more output values being produced. Under appropriate technical conditions on the ordered sets associated to the communication channels, and the functions modeling the computations performed by the system components, this results in a well behaved framework, where components can be connected together, even forming loops, and always resulting in a unique and well defined function describing the input/output behavior of the whole system. Previous approaches to model interactive systems, such as Kahn networks [16] and Maurer’s random systems [21], can indeed be seen as special cases of our general process model.¹ The resulting model is quite powerful, allowing even to model probabilistic computation as a special case. However, the simplicity of the model has a price: all components of the system must be monotone with respect to the information ordering relation. For example, if a program P on input messages x_1, x_2 outputs $P(x_1, x_2) = (y_1, y_2, y_3)$, then on input x_1, x_2, x_3 it can only output a sequence of messages that extends (y_1, y_2, y_3) with more output. In other words, P cannot “go back in time” and change y_1, y_2, y_3 . While this is a very natural and seemingly innocuous restriction, it also means that the program run by P cannot perform operations of the form “if no input message has been received yet, then send y ”. This is because if an input message is received at a later point, P cannot go back in time and not send y .

It is our thesis that these time dependent operations make cryptographic protocols harder to understand and analyze, and therefore should be avoided whenever possible.

Organization. The rest of the paper is organized as follows. In Section 2 we present our framework for the description and analysis of concurrent processes, and illustrate the definitions using a toy example. Next, we demonstrate the applicability of the framework by carefully describing and analyzing two classic cryptographic protocols: secure broadcast (in Section 3) and verifiable secret sharing (in Section 4). The secure broadcast protocol in Section 3 is essentially the one of Bracha, and only uses deterministic functions. Our modular analysis of the protocol illustrates the use of subprotocols that are not universally composable. The verifiable secret sharing protocol analyzed in Section 4 provides an example of randomized protocol.

¹For the readers well versed in the subject, we remark that our model can be regarded as a generalization of Kahn networks where the channel behaviors are elements of arbitrary partially ordered sets (or, more precisely, domains) rather than simple sequences of messages. This is a significant generalization that allows to deal with probabilistic computations and intrinsically nondeterministic systems seamlessly, without incurring into the Brock-Ackerman anomaly and similar problems.

2 Distributed Systems, Composition, and Secure Computation

In this section we introduce our mathematical framework for the description and analysis of distributed systems. We start with a high level description of our approach, which will be sufficient to apply our framework and to follow the proofs. We then give more foundational details justifying soundness of our approach. Finally, we provide security definitions for protocols in our framework.

2.1 Processes and systems

Introducing processes: An example and notational conventions. Our framework models (asynchronous and reactive) processes and systems with one or more input and output *channels* as mathematical functions mapping input histories to output histories. Before introducing more formal definitions, let us illustrate this concept with a simple example. Consider a deterministic process with one input and one output channels, which receives as input a sequence of messages, $x[1], \dots, x[k]$, where each $x[k]$ is (or can be parsed as) an integer. The process receives the messages sequentially, one at a time, and in order to make the process finite one may assume that the process will accept only the first n messages. Upon receiving each input message $x[i]$, the process increments the value and immediately outputs $x[i] + 1$. It is not hard to model the process in terms of a function mapping input to output sequences: The input and output of the function modeling the process are the set $\mathbb{Z}^{\leq n}$ of integer sequences of length at most n , and the process is described by the function $F: \mathbb{Z}^{\leq n} \rightarrow \mathbb{Z}^{\leq n}$ mapping each input sequence $x \in \mathbb{Z}^k$ (for some $k \leq n$) to the output sequence $y \in \mathbb{Z}^k$ of the same length defined by the equations $y[i] = x[i] + 1$ (for $i = 1, \dots, k$). There are multiple ways one can possibly describe such a function. We describe the process in equational form as in Figure 6 (left). In the example, the first line assigns names to the function, input and output variables, while the remaining lines are equations that define the value of the output variables in terms of the input variables. Each variable ranges over a specific set ($x, y \in \mathbb{Z}^{\leq n}$), but for simplicity we often leave the specification of this set implicit, as it is usually clear from the context. By convention, all variables that appear in the equations, but not as part of the input or output variables, are considered *local/internal* variables, whose only purpose is to help defining the value of the output variables in terms of the input variables. Free index variables (e.g., i, j) are universally quantified (over appropriate ranges) and used to compactly describe sets of similar equations.

Processes as monotone functions. In general, the reason we define a process F as a function mapping sequences to sequences² (rather than, say, as a function $f(x) = x + 1$ applied to each incoming message x) is that it allows to describe the most general type of (e.g., stateful, reactive) process, whose output is a function of all messages received as input during the execution of the protocol. (Note that we do not need to model state explicitly.) Also, such functions can describe processes with multiple input and output channels by letting inputs and outputs be *tuples* of message sequences. However, clearly, not any such function mapping input to output sequences can be a valid process. To capture valid functions representing a process, input and output sets are endowed with a partial ordering relation \leq , where $x \leq y$ means that y is a possible *future* of x . (In the case of sequences of messages, \leq is the standard prefix partial ordering relation, where $x \leq y$ if $y = x|z$ for some other sequence z , and $x|z$ is the concatenation of the two sequences.) Functions

²Here we use sequences just as a concrete example. Our framework uses more general structures, namely domains.

$$\left| \begin{array}{l} F(x) = y: \\ y[i] = x[i] + 1 \quad (i = 1, \dots, |x|) \end{array} \right| \left| \begin{array}{l} G(y) = (z, w): \\ z = y \\ w = y \end{array} \right| \left| \begin{array}{l} H(z) = x: \\ x[1] = 1 \\ x[j + 1] = z[j] \quad (j \leq \min\{|z|, n - 1\}) \end{array} \right|$$

Figure 6: Some simple processes

describing processes should be naturally restricted to monotone functions, i.e., functions such that $x \leq y$ implies $F(x) \leq F(y)$. In our example, this simply means that if on input a sequence of messages x , $F(x)$ is produced as output, upon receiving additional messages z , the output sequence can only get longer, i.e., $F(y) = F(x|z) = F(x)|z'$ for some z' . In other words, once the messages $F(x)$ are sent out, the process cannot change its mind and set the output to a sequence that does not start with $F(x)$.

Note that so far we only discussed an example of a *deterministic* process. Below, after introducing some further foundational tools, we will see that probabilistic processes are captured in the same way by letting the function output be a *distribution* over sequences, rather than a single sequence of symbols.

Further examples and notational conventions. In the examples, $|x|$ denotes the length of a sequence x , and we use array notation $x[i]$ to index the elements of a sequence. Figure 6 gives two more examples of processes that further illustrate notational conventions. Process G, in Figure 6 (middle), simply duplicates the input y (as usual in $\mathbb{Z}^{\leq n}$) and copies the input messages to two different output channels z and w . When input or output values are tuples, we usually give separate names to each component of the tuple. As before, all variables take values in $\mathbb{Z}^{\leq n}$ and the output values are defined by a set of equations that express the output in terms of the input. Finally, process H(z) takes as input a sequence $z \in \mathbb{Z}^{\leq n}$, and outputs the message 1 followed by the messages z received as input, possibly truncated to a prefix $z[< n]$ of length at most $n - 1$, so that the output sequence x has length at most n .

Process composition. Processes are *composed* in the expected way, connecting some output variables to other input variables. Here we use the convention that variable *names* are used to implicitly specify how different processes are meant to be connected together.³ Composing two processes together yields, in turn, another process, which is obtained simply combining all the equations. We often refer to the resulting process as a *system* to stress its structure as a composition of basic processes. However, it should be noted that both a process and a system are objects of the same mathematical type, namely monotone functions described by systems of equations. For example, the result of composing G and H from Figure 6 yields the process $(G \mid H)$ shown in Figure 7 (left), with input y and output (w, x) , where $w = y$ replicates the input to make it externally visible. We use the convention that by default processes are connected by private channels, not visible outside of the system. This is modeled by turning their common input/output variables into local ones, not part of the input or output of the composed system. Of course, one can always either override this convention by explicitly listing such common input/output variables as part of the output, or bypass it by duplicating the value of a variable as done for example by process G.

³We stress that this is just a notational convention, and there are many other syntactical mechanisms that can be used to specify the “wiring” in more or less explicit ways.

$$\left| \begin{array}{l} [G \mid H](y) = (w, x): \\ z = y \\ w = y \\ x[1] = 1 \\ x[j+1] = z[j] \end{array} \right| (j < n) \quad \left| \begin{array}{l} [G \mid H](y) = (w, x): \\ w = y \\ x[1] = 1 \\ x[j+1] = y[j] \end{array} \right| (j < n)$$

Figure 7: Process composition

This is just a syntactical convention, and several other choices are possible, including never hiding variables during process composition and introducing a special projection operator to hide internal variables.

Since processes formally define functions (from input to output variables), and equations are just a syntactic method to specify functions, equations can be simplified without affecting the process. Simplifications are easily performed by substitution and variable elimination. For example, using the first equation $z = y$, one can substitute y for z , turning the last equation in the system into $x[i+1] = y[i]$. At this point, the local variable z is no longer used anywhere, and its defining equation can be removed from the system. The result is shown in Figure 7 (right). We remark that the two systems of equations shown in Figure 7 define the *same* process: they have the same input and output variables, and the equations define precisely the same function.

Feedback loops and recursive equations. Now consider the composition of all three processes F,G,H from Figure 6. Composition can be performed one pair at a time, and in any order, e.g., as $[[F \mid G] \mid H]$ or $[F \mid [G \mid H]]$. Given the appropriate mathematical definitions, it can be easily shown that the result is the same, independent from the order of composition. (This is clear at the syntactic level, where process composition is simply defined by combining all the equations together. But associativity of composition can also be proved at the semantic level, where the objects being combined are functions.) So, we write $[F \mid G \mid H]$ to denote the result of composing multiple processes together, shown in Figure 8 (left). When studying multi-party computation protocols, one is naturally led to consider collections of processes, e.g., P_1, \dots, P_n , corresponding to the individual programs run by each participant. Given a collection $\{P_i\}_i$ and a subset of indices $I \subseteq \{1, \dots, n\}$, we write P_I to denote the composition of all P_i with $i \in I$. Similarly, we use x_A or $x[A]$ to denote a vector indexed by $i \in A$. As a matter of notation, we also use x^A to denote the $|A|$ -dimensional vector indexed by $i \in A$ with all components set equal to x .

The system $[F \mid G \mid H]$ has no inputs, and only one output w . More interestingly, the result of composing all three processes yields a recursive system of equations, where y is a function of x , x is a function of z and z is a function of y . Before worrying about solving the recursion, we can simplify the system. A few substitutions and variable eliminations yield the system in Figure 8 (right). The system consists of a single, recursively defined output variable w . The recursive definition of w is easy to solve, yielding $w[i] = i + 1$ for $i \leq n$.

2.2 Foundations: Domain theory and probabilistic processes

So far, equations have been treated in an intuitive and semi-formal way, and in fact obtaining an intuitive and lightweight framework is one of our main objectives. But for the approach to be sound, it is important that the equations and the variable symbols manipulated during the design

$$\left| \begin{array}{l} [F \mid G \mid H]() = w: \\ y[i] = x[i] + 1 \\ z = y \\ w = y \\ x[1] = 1 \\ x[j+1] = z[j] \end{array} \right| \begin{array}{l} (i \leq n) \\ \\ \\ \\ (j < n) \end{array} \left| \begin{array}{l} [F \mid G \mid H]() = w: \\ w[1] = 2 \\ w[j+1] = w[j] + 1 \end{array} \right| \begin{array}{l} \\ \\ (j < n) \end{array}$$

Figure 8: Example of recursive process

and analysis of a system be given a precise mathematical meaning. Also, we want to consider a more general model of processes where inputs and outputs are not restricted to simple sequences of messages, but can be more complex objects, including probability distributions. This requires us to introduce some further formal tools.

The standard framework to give a precise meaning to our equations is that of *domain theory*, a well established area of computer science developed decades ago to give a solid foundation to functional programming languages [27, 26, 14, 28, 1]. Offering a full introduction to domain theory is beyond the scope of this paper, but in order to reassure the reader that our framework is sound, we recall the most basic notions and illustrate how they apply to our setting.

Domains and partial orders. *Domains* are a special kind of partially ordered set satisfying certain technical properties. We recall that a partially ordered set (or poset) $(X; \leq)$ is a set X together with a reflexive, transitive and antisymmetric relation \leq . We use posets to model the set of possible histories (or behaviors) of communication channels, with the partial order relation corresponding to temporal evolution. For example, a channel that allows the transmission of an arbitrary number of messages from a basic set M (and that preserves the order of transmitted messages) can be modeled by the poset $(M^*; \leq)$ of finite sequences of elements of M together with the prefix partial ordering relation \leq . A *chain* $x_1 \leq x_2 \leq \dots \leq x_n$ represents a sequence of observations at different points in time.⁴ In this paper we will extensively use an even simpler poset M_\perp , consisting of the base set M extended with a special “bottom” element \perp , and the flat partial order where $x \leq y$ if and only if $x = \perp$ or $x = y$. The poset M_\perp is used to model a communication channel that allows the transmission of a single message from M , with the special value \perp representing a state in which no message has been sent yet.

The Scott topology and continuity. Posets can be endowed with a natural topology, called the *Scott topology*, that plays an important role in many definitions. In the case of posets $(X; \leq)$ with no infinite chains, closed sets can be simply defined as sets $C \subseteq X$ that are downward closed, i.e., if $x \in C$ and $y \leq x$, then $y \in C$. Intuitively, a set is closed if it contains all possible “pasts” that lead to a current set of events. Open sets are defined as usual as the complements of closed sets. It is easy to see that the standard (topological⁵) definition of continuous function $f: X \rightarrow Y$ (according to the Scott topology on posets with no infinite chains) boils down to requiring that f is monotone, i.e., for all $x, y \in X$, if $x \leq y$ in X , then $f(x) \leq f(y)$ in Y . In the case of posets with

⁴Domain theory usually resorts to the (related, but more general) notion of *directed set*. But not much is lost by restricting the treatment to chains, which are perhaps more intuitive to use in our setting.

⁵We recall that a function $f: X \rightarrow Y$ between two topological spaces is continuous if the preimage $f^{-1}(O)$ of any open set $O \subset Y$ is also open.

infinite chains, such as $(M^*; \leq)$, definitions are slightly more complex, and require the definition of limits of infinite chains. For any poset $(X; \leq)$ and subset $A \subseteq X$, $x \in X$ is an upper bound on A if $x \geq a$ for all $a \in A$. The value x is called the least upper bound of A if it is an upper bound on A , and any other upper bound y satisfies $x \leq y$. Informally, if $A = \{a_i \mid i = 1, 2, \dots\}$ is a chain $a_1 \leq a_2 \leq a_3 \leq \dots$, and A admits a least upper bound (denoted $\bigvee A$), then we think of $\bigvee A$ as the limit of the monotonically increasing sequence A . (In our setting, where the partial order models temporal evolution, the limit corresponds to the value of the variable describing the entire channel history once the protocol has finished executing.) All Scott domains (and all posets used in this paper) are complete partial orders (or CPO), i.e., posets such that all chains $A \subseteq X$ admit a least upper bound. CPOs have a minimal element $\perp = \bigvee \emptyset$, which satisfies $\perp \leq x$ for all $x \in X$. Closed sets $C \subseteq X$ of arbitrary CPOs X are defined by requiring C to be also closed under limits, i.e., for any chain $Z \subseteq C$ it must be $\bigvee Z \in C$. (Open sets are always defined as the complement of closed sets.) Similarly, continuous functions between CPOs $f: X \rightarrow Y$ should preserve limits, i.e., any chain $Z \subseteq X$ must satisfy $f(\bigvee Z) = \bigvee f(Z)$.

As an example, we see that $(M^*; \leq)$ is not a CPO. We can define infinite chains A of successively longer strings (e.g., take $x_i = 0^i$ for $M = \{0, 1\}$) such that no limit in M^* exists for this chain. However, note that such a chain always defines an infinite string $x^* \in M^\infty$ which is such that $x^* \leq y$ holds for all $A \leq x$. Therefore, the poset $(M^* \cup M^\infty; \leq)$ is a CPO.⁶ This CPO can be used to model processes taking input and output sequences of arbitrary length.

Later on, we often use generalizations of the above limit notion, called the *join* and the *meet*, respectively. For a set $Z \subseteq X$, let $Z^\uparrow = \{z' \in X : \forall z \in Z : z \leq z'\}$ the set of upper bounds on Z . An element $z^* \in Z^\uparrow$ such that $z^* \leq z$ for all $z \in Z^\uparrow$, if it exists, is called the *join* of Z and denoted $\bigvee Z$. The set Z^\downarrow and the *meet* $\bigwedge Z$ are defined symmetrically.

Equational descriptions and fixed points. We can now provide formal justification for our equational approach given above. Note that CPOs can be combined in a variety of ways, using common set operations, while preserving the CPO structure. For example, the cartesian product $A \times B$ of two CPOs is a CPO with the component-wise partial ordering relation. Using cartesian products, one can always describe every valid system of equations (as informally used in the previous paragraphs to define a process or a system) as the definition of a function f of the form

$$f(z) = g(z, x) \quad \text{where} \quad x = h(z, x) \quad (1)$$

for some internal variable x and bivariate continuous⁷ functions $h(z, x)$ and $g(z, x)$. An important property of CPOs is that every continuous function $f: X \rightarrow X$ admits a least fixed point, i.e., a minimal $x \in X$ such that $f(x) = x$, which can be obtained by taking the limit of the chain $\perp \leq f(\perp) \leq \dots \leq f^n(\perp) \leq \dots$, and admits an intuitive operational interpretation: starting from the initial value $x = \perp$, one keeps updating the value $x \leftarrow f(x)$ until the computation stabilizes.

Least fixed points are used to define the solution to recursive equations as (1) above as follows, and to show that it is always defined, proving soundness of our approach. For any fixed z , the function $h_z(x) = h(z, x)$ is also continuous, and maps X to itself. So, it admits a least fixed point $x_z = \bigvee_i h_z^i(\perp)$. The function defined by (1) is precisely $f(z) = g(z, x_z)$ where x_z is the least fixed point of $h_z(x) = h(z, x)$. It is a standard exercise to show that the function $f(z)$ so defined is a continuous function of z .

⁶In fact, usually, one can define M^∞ to be exactly the set of limits of infinite chains from M^* .

⁷Continuity for bivariate functions is defined regarding f as an univariate function with domain $Z \times X$.

Scott domains are a special class of CPOs satisfying a number of additional properties (technically, they are algebraic bounded complete CPOs), that are useful for the full development of the theory. As most of the concepts used in this paper can be fully described in terms of CPOs, we do not introduce additional definitions, and refer the reader to any introductory textbook on domain theory for a formal treatment of the subject.

Probabilistic processes. So far, our theory does not support yet the definition of processes with probabilistic behavior. Intuitively, we want to define a process as a continuous map from elements of a CPO X to probability *distributions* over some CPO Y . We will now discuss how to define the set $D(Y)$ of such probability distributions, which turns out to be a CPO. Our approach follows [25].

Let $O(X)$ be the open sets of X , and $B(X)$ the Borel algebra of X , i.e., the smallest σ -algebra that contains $O(X)$. We recall that a probability distribution over a set X is a function $p: B(X) \rightarrow [0, 1]$ that is countably additive and has total mass $p(X) = 1$. The set of probability distributions over a CPO X , denoted $D(X)$, is a CPO according to the partial order relation such that $p \leq q$ if and only if $p(A) \leq q(A)$ for all open sets $A \in O(X)$. This partial order on probability distributions $D(X)$ captures precisely the natural notion of evolution of a probabilistic process: the probability of a closed set can only decrease as the system evolves and probability mass “escapes” from it into the future. A probabilistic process P with input in X and output in Y is described by a continuous function from X to $D(Y)$ that on input an element $x \in X$ produces an output probability distribution $P(x) \in D(Y)$ over the set Y .

While these mathematical definitions may seem somehow arbitrary and complicated, we reassure the reader that they correspond precisely to the common notion of probabilistic computation. For example, any function $P: X \rightarrow D(Y)$ can be uniquely extended to take as input probability distributions. The resulting function $\hat{P}: D(X) \rightarrow D(Y)$, on input a distribution D_X , produces precisely what one could expect: the output probability distribution $D_Y = \hat{P}(D_X)$ is obtained by first sampling $x \leftarrow D_X$ according to the input distribution, and then sampling the output according to $y \leftarrow P(x)$. Moreover, the result $\hat{P}: D(X) \rightarrow D(Y)$ is continuous according to the standard topology of $D(X)$ and $D(Y)$.

The fact that a distribution $D_X \in D(X)$ and a function $f: X \rightarrow D(Y)$ can be combined to obtain an output distribution D_Y allows to extend our equational treatment of systems to probabilistic computations. A probabilistic system is described by a set of equations similar to (1), except that h is a continuous function from $Z \times X$ to $D(X)$, and we write the equation in the form $x \leftarrow h(z, x)$ to emphasize that $h(z, x)$ is a probability distribution to sample from, rather than a single value. For any fixed z , the function $h_z(x) = h(z, x)$ is continuous, and it can be extended to a continuous function $\hat{h}_z: D(X) \rightarrow D(X)$. The least fixed point of this function is a probability distribution $D_z \in D(X)$, and function f maps the value z to the distribution $g(z, D_z)$.

Formally, the standard mathematical tool to give the equations a precise meaning is the use of monads, where \leftarrow corresponds to the monad “bind” operation. We reassure the reader that this is all standard, well studied in the context of category theory and programming language design, both in theory and practice, e.g., as implemented in mainstream functional programming languages like Haskell. Rigorous mathematical definitions to support the definition of systems of probabilistic equations can be easily given within the framework of domain theory, but no deep knowledge of the theory is necessary to work with the equations, just like knowledge of denotational semantics is not needed to write working computer programs.

2.3 Multi-party computation, security and composability

So far, we have developed a domain-theoretic framework to define processes, their composition, and their asynchronous interaction. We still need to define what it means for such a system to implement a multi-party protocol, and what it means for such a protocol to securely implement some functionality. Throughout this section, we give definitions in the deterministic case for simplicity. The definitions extend naturally to probabilistic processes by letting the output being a probability distribution over (the product of) the output sets.

We model secure multi-party computation along the lines described in the introduction. A secure computation task is modeled by an n -party functionality F that maps n inputs (x_1, \dots, x_n) to n outputs (y_1, \dots, y_n) in the deterministic case, or to a distribution on a set of n outputs in the probabilistic case. Each input or output variable is associated to a specific domain X_i/Y_i , and F is a continuous function $F: (X_1 \times \dots \times X_n) \rightarrow (Y_1 \times \dots \times Y_n)$, typically described by a system of domain equations. Each pair X_i/Y_i corresponds to the input and output channels used by user i to access the functionality. We remark that, within our framework, even if F is a (pure) mathematical function, it still models a reactive functionality that can receive inputs and produce outputs asynchronously in multiple rounds.

Sometimes, one knows in advance that F will be used within a certain context. (For example, in the next section we will consider a multicast channel that is always used for broadcast, i.e., in a context where the set of recipient is always set to the entire group of users.) In these settings, for efficiency reasons, it is useful to consider protocols that do not implement the functionality F directly, but only the use of F within the prescribed context. We formalize this usage by introducing the notion of a protocol implementing an *interface* to a functionality. An interface is a collection of continuous functions $I_i: X'_i \times Y_i \rightarrow X_i \times Y'_i$, where X'_i, Y'_i are the input and output domain of the interface. Combining the interface $I = I_1 \mid \dots \mid I_n$ with the functionality F , yields a system $(F \mid I)$ with inputs X'_1, \dots, X'_n and outputs Y'_1, \dots, Y'_n that offers a limited access to F . The standard definition of (universally composable) security corresponds to setting I to the trivial interface where $X'_i = X_i$, $Y'_i = Y_i$ and each I_i to the identity function offering direct access to F .

Ideal functionalities can be used both to describe protocol problems, and underlying communication models. Let $N: S_1 \times \dots \times S_n \rightarrow R_1 \times \dots \times R_n$ be an arbitrary ideal functionality. One may think of N as modeling a communication network where user i sends $s_i \in S_i$ and receives $r_i \in R_i$, but all definitions apply to arbitrary N .

A protocol implementing an interface I to functionality F in the communication model N is a collection of functions P_1, \dots, P_n where $P_i: X'_i \times R_i \rightarrow Y'_i \times S_i$. We consider the execution of protocol P in a setting where an adversary can corrupt a subset of the participants. The set of corrupted players $A \subseteq \{1, \dots, n\}$ must belong to a given family \mathcal{A} of allowable sets, e.g., all sets of size less than $n/2$ in case security is to be guaranteed only for honest majorities. We can now define security.

Definition 1 *Protocol P securely implements interface I to functionality F in the communication model N if for any allowable set $A \in \mathcal{A}$ and complementary set $H = \{1, \dots, n\} \setminus A$, there is a simulator $S: S_A \times Y_A \rightarrow X_A \times R_A$ such that the systems $(P_H \mid N)$ and $(S \mid I_H \mid F)$ are equivalent, i.e., they define the same function.*

$(P_H \mid N)$ is called the real system, and corresponds to an execution of the protocol in which the users in A are corrupted, while those in H are honest and follow the protocol. It is useful to

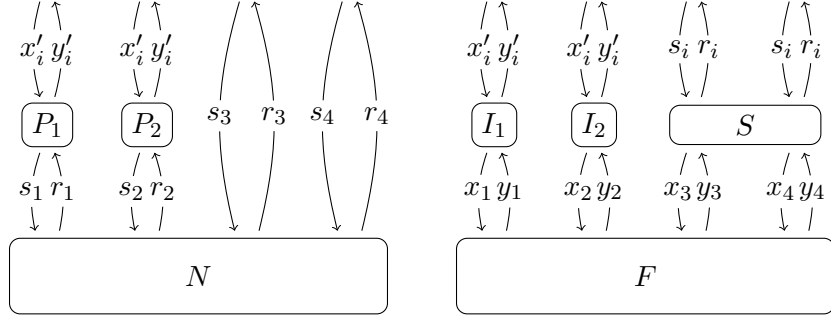


Figure 9: The protocol (P_1, \dots, P_4) securely implements interface (I_1, \dots, I_4) to functionality F in the communication model N .

illustrate this system with a diagram. See Figure 9 (left). We see from the diagram that the real system as inputs X'_H, S_A and outputs Y'_H, R_A . In the ideal setting, when the adversary corrupts the users in A , we are left with the system $I_H \mid F$ because corrupted users are not bound to use the intended interface I . This system $I_H \mid F$ has inputs X'_H, X_A and outputs Y'_H, Y_A . In order to turn this system into one with the same inputs and outputs as the real one, we need a simulator of type $S: S_A \times Y_A \rightarrow X_A \times R_A$. When we compose S with $I_H \mid F$ we get a system $(S \mid I_H \mid F)$ with the same input and output variables as the real system $(P_H \mid N)$. See Figure 9 (right). For the protocol to be secure, the two systems must be equivalent, showing that any attack that can be carried out on the real system by corrupting the set A can be simulated on the ideal system through the simulator.

When composing protocols together, N is not a communication network, but an ideal functionality representing a hybrid model. In this setting, we say that protocol P accesses N through interface $J = (J_1, \dots, J_n)$ if each party runs a program of the form $P_i = J_i \mid P'_i$. If this is the case, we say that P securely implements interface I to functionality F through interface J to communication model N .

Composition theorems in our framework come essentially for free, and their proof easily follow from the general properties of systems of equations. For example, we have the following rather general composition theorem.

Theorem 1 *Assume P securely implements interface I to F in the communication model N , and $Q = Q' \mid I$ securely implements G through interface I to F , then the composed protocol $Q' \mid P$ securely implements G in the communication model N .*

The simple proof is similar to the informal argument presented in the introduction, and it is left to the reader as an exercise. The composition theorem is easily extended in several ways, e.g., by considering protocols Q that only implement a given interface J to G , and protocols P that use N through some given interface J' .

$\text{BCAST}(x) = (y_1, \dots, y_n):$ $y_i = x \quad (i = 1, \dots, n)$	$\text{DEALER}(x) = (x', w):$ $w[i] = \top \quad (i = 1, \dots, n)$ $x' = x$
$\text{WCAST}(x', w) = (y'_1, \dots, y'_n):$ $y'_i = x' \wedge w[i] \quad (i = 1, \dots, n)$	$\text{PLAYER}[i](y'_i, r_i) = (y_i, s_i): \quad (i = 1, \dots, n)$ $s_i[j] = y'_i \vee t_1(r_i[1], \dots, r_i[n]) \quad (j = 1, \dots, n)$ $y_i = t_2(r_i[1], \dots, r_i[n])$
$\text{NET}(s_1, \dots, s_n) = (r_1, \dots, r_n):$ $r_i[j] = s_j[i] \quad (i, j = 1, \dots, n)$	

Figure 10: The Broadcast protocol

3 Secure Broadcast

In this section we provide, as a simple case study, the analysis of a secure broadcast protocol (similar to Bracha's reliable broadcast protocol [8]), implemented on an asynchronous point-to-point network. We proceed in two steps. In the first step, we build a weak broadcast protocol, that provides consistency, but does not guarantee that all parties terminate with an output. In the second step, we use the weak broadcast protocol to build a protocol achieving full security. We present the two steps in reverse order, first showing how to strengthen a weak broadcast protocol, and then implementing the weak broadcast on a point-to-point network.

3.1 Building broadcast from weak broadcast

In this section we build a secure broadcast protocol on top of a weak broadcast channel and a point to point communication network. The broadcast, weak broadcast, and communication network are described in Figure 10 (left). The broadcast functionality (BCAST) receives a message x from a dealer, and sends a copy $y_i = x$ to each player. The weak broadcast channel (WCAST) allows a dishonest dealer to specify (using a boolean vector $w \in \{\perp, \top\}^n$) which subset of the players will receive the message. Notice that the functionality WCAST described in Figure 10 is in fact a multicast channel, that allows the sender to transmit a message to any subset of players of its choice. We call it a weak broadcast, rather than multicast, because we will not use (or implement) this functionality at its full power: the honest dealer in our protocol will always set all $w_i = \top$, and use WCAST as a broadcast channel $\text{BCAST}(x) = \text{WCAST}(x, \top^n)$. The auxiliary inputs w_i are used only to capture the extra power given to a dishonest dealer that, by not following the protocol, may restrict the delivery of the message x to a subset of the players. This will be used in the next section to provide a secure implementation of WCAST on top of a point to point communication network.

The broadcast protocol is very simple and it is shown in Figure 10 (right). The dealer simply uses WCAST to transmit its input message x to all n players by setting $w[i] = \top$ for all $i \in [n]$. The players have then access to a network functionality NET to exchange messages among themselves. The program run by the players makes use to two threshold functions t_1 and t_2 each taking n inputs, which are assumed to satisfy, for every admissible set of corrupted players $A \subseteq \{1, \dots, n\}$ and complementary set $H = \{1, \dots, n\} \setminus A$, input vector u , and value x , the following properties: $t_1(u[A], (x)^H) = t_2(u[A], (x)^H) = x$ (i.e., if all honest players agree on x , then t_1, t_2 output x irrespective of the other values), and $t_1((\perp)^A, u[H]) \geq t_2((\top)^A, u[H])$ (i.e., for any set of values

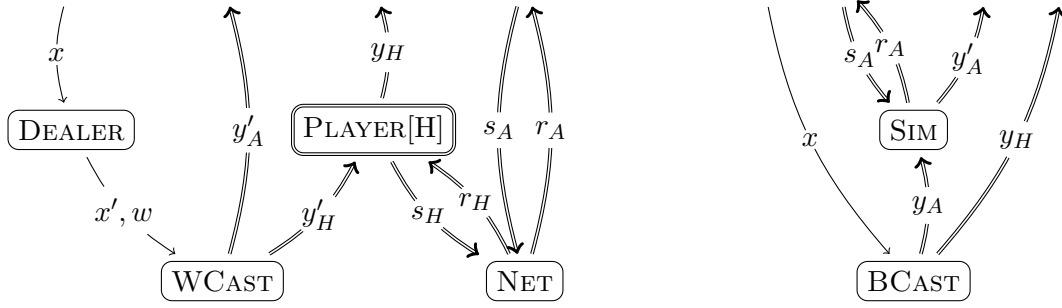


Figure 11: Security of broadcast protocol when the dealer is honest

provided by the honest players, t_1 is always bigger than t_2 regardless of the other values). It is easy to see that the threshold functions $t_i(u) = \bigvee_{|S|=k_i} \bigwedge_{j \in S} u_j$ satisfy these properties provided $|A| < k_1 \leq k_2 - |A| \leq n - 2|A|$, which in particular requires $n \geq |A| + 1$.

In the security analysis, we distinguish two cases, depending on whether the dealer is corrupt or not.

Honest dealer. First we consider the simple case where the adversary corrupts a set of players $A \subset \{1, \dots, n\}$, and the dealer behaves honestly. Let $H = \{1, \dots, n\} \setminus A$ be the set of honest players. An execution of the protocol when players in A are corrupted is described by the system (DEALER | PLAYER[H] | WCAST | NET) with input (x, s_A) and output (y_H, y'_A, r_A) depicted in Figure 11 (left). Note that in this (and the following) figures, double arrows and boxes denote parallel processes and channels. Combining the defining equations of DEALER, PLAYER[h] for $h \in H$, WCAST and NET, and introducing the auxiliary variables $u_h = x \vee t_1(s_A[h], s_H[h])$ for all $h \in H$, we get that for any $i, j \in [n]$, and $h \in H$ the following holds:

$$\begin{aligned}
 r_j[i] &= s_i[j] \\
 y'_i &= x' \wedge w[i] = x \wedge \top = x \\
 s_h[i] &= y'_h \vee t_1(r_h[1], \dots, r_h[n]) = x \vee t_1(s_A[h], s_H[h]) = u_h \\
 y_h &= t_2(r_h[1], \dots, r_h[n]) = t_2(s_A[h], s_H[h]) = t_2(s_A[h], u_H) \\
 u_h &= x \vee t_1(s_A[h], s_H[h]) = x \vee t_1(s_A(h), u_H) .
 \end{aligned}$$

The last equation $u_h = x \vee t_1(s_A(h), u_H)$ provides a recursive definition of u_H , which can be easily solved by an iterative least fix point computation: starting from $u_H^{(0)} = \perp^H$, we get $u_H^{(1)} = (x \vee t_1(s_A(h), u_H^{(0)}))^H = x^H$, and then again $u_H^{(2)} = (x \vee t_1(s_A(h), u_H^{(1)}))^H = x^H$. Therefore the least fix point is $u_H = x^H$. Substituting $u_H = x^H$ in the previous equations, and using the properties of t_2 , we see that the system of equations defined by (DEALER | PLAYER[H] | WCAST | NET) is equivalent to

$$\begin{aligned}
 r_a &= (s_A[a], x^H) & (a \in A) \\
 y'_a &= x & (a \in A) \\
 y_h &= t_2(s_A[h], x^H) = x & (h \in H)
 \end{aligned} \tag{2}$$

We now show that an equivalent system can be obtained by combining the ideal functionality BCAST with a simulator SIM as in Figure 11 (right). The simulator takes (y_A, s_A) as input, and

$\text{SIM}(y_A, s_A) = (y'_A, r_A):$ $\begin{aligned} r_A[a] &= s_A[A] & (a \in A) \\ r_A[h] &= y_A & (h \in H) \\ y'_A &= y_A \end{aligned}$	$\text{SIM}'(x', w, y_A, s_A) = (x, r_A, y'_A):$ $\begin{aligned} u_h &= (x' \wedge w[h]) \vee t_1(s_A[h], u_H) & (h \in H) \\ x &= t_2(s_A[h], u_H) & (h = \min H) \\ y'_a &= x' \wedge w[a] & (a \in A) \\ r_a[A] &= s_A[a] & (a \in A) \\ r_a[H] &= u_H & (a \in A) \end{aligned}$
------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Figure 12: Simulators for the broadcast protocol when the dealer is honest (left) or dishonest (right)

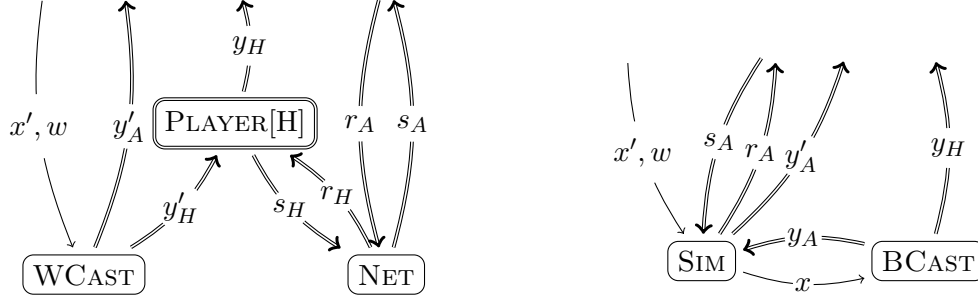


Figure 13: Security of broadcast protocol when the dealer is corrupted.

must output (y'_A, r_A) such that $(\text{SIM} \mid \text{BCAST})$ is equivalent to the system $(\text{DEALER} \mid \text{PLAYER}[H] \mid \text{WCAST} \mid \text{NET})$ specified by the last set of equations. The simulator is given in Figure 12 (left). It is immediate to verify that combining the equations of the simulator SIM with the equations $y_i = x$ of the ideal broadcast functionality, and eliminating local variables, yields a system of equations identical to (2).

Dishonest dealer. We now consider the case where both the dealer and a subset of players A are corrupted. As before, let $H = \{1, \dots, n\} \setminus A$ be the set of honest players. The system corresponding to a real execution of the protocol when DEALER and $\text{PLAYER}[A]$ are corrupted is $(\text{PLAYER}[H] \mid \text{WCAST} \mid \text{NET})$, mapping (x', w, s_A) to (y_H, r_A, y'_A) . (See Figure 13 (left).) Using the defining equations of $\text{PLAYER}[H]$, WCAST and NET , and introducing auxiliary variables $u_h = y'_h \vee t_1(r_h[1], \dots, r_h[n])$ for $h \in H$, we get the following set of equations:

$$\begin{aligned} y_h &= t_2(r_h[A], r_h[H]) = t_2(s_A[h], u_H) & (h \in H) \\ y'_a &= x' \wedge w[a] & (a \in A) \\ r_a[A] &= s_A[a] & (a \in A) \\ r_a[H] &= u_H \\ u_h &= (x' \wedge w[h]) \vee t_1(s_A[h], u_H) & (h \in H) \end{aligned} \tag{3}$$

This time the simulator SIM' takes input (x', w, y_A, s_A) and outputs (x, r_A, y'_A) . (See Figure 13 (right).) With these inputs and outputs, the simulator can directly set all variables except y_h just as in the real system (3). The simulator can also compute the value y_h , but it cannot set y_h directly because this variable is defined by the ideal functionality as $y_h = x$. We will prove that all variables y_h defined by (3) take the same value. It follows that the simulator can set $x = y_h$ for any $h \in H$,

and the system (BCAST, SIM') will be equivalent to (3) (and therefore to (PLAYER[H] | WCAST | NET)). The code of the simulator is given in Figure 12 (right), where $x = y_h$ is arbitrarily selected using the smallest index $h = \min H$. (Any other choice of h would have been fine.)

It remains to prove that all y_h take the same value. By antisymmetry, it is enough to show that $y_i \leq y_j$ for all $i, j \in H$. This easily follows from the assumptions on t_1, t_2 . In fact, by monotonicity, we have

$$y_i = t_2(s_A[i], u_H) \leq t_2(\top^A, u_H) \leq t_1(\perp^A, u_H) \leq t_1(s_A[j], u_H) \leq y_j.$$

It immediately follows that $y_j = t_2(s_A[j], u_H) \geq t_2(s_A[j], (y_i)^H) = y_i$.

3.2 Weak broadcast

In this section we show how to implement the weak broadcast functionality WCAST given in Figure 10 to be used within the BCAST protocol discussed in the previous section, and analyze its security. We recall that WCAST is a multicast functionality connecting a dealer to n other parties, which allows the dealer to send a message x' to a subset of the parties specified by a vector $w \in \{\perp, \top\}^n$. We stress that we do not need a secure implementation of WCAST in its full generality, as our higher level broadcast protocol (BCAST) uses WCAST in a rather restricted way: it always set $w = (\top)^n$ and transmits x' to all parties. Accordingly, we give a protocol that securely implements this interface to WCAST. Formally, the dealer's interface INT takes only x' as external input, and passes it along with $w = \top^n$ to the ideal functionality. The other parties have unrestricted access to the ideal functionality, and their interface is the identity function (or empty system of equations).

We implement interface INT to WCAST on top of a point-to-point communication network similar to the NET functionality described in Figure 10, with the only difference that here also the dealer can send messages. The protocol is very simple: the dealer transmits the input x' to all parties, and the parties retransmit the message to each other. Each party sets its output using a threshold function of the messages received by the other parties. The equations corresponding to the network NET', interface INT, and protocol programs DEALER, PLAYER[1], ..., PLAYER[n] are given in Figure 14. For reference, we have also repeated the definition of BCAST from Figure 10. The function t is assumed to satisfy the following properties: $t(u[A], (x)^H) = x$ (i.e., if all honest parties agree on x , then the output is x), and, moreover, for all vectors u, u' with $u[H] = u'[H]$, we have $t(u) = t(u')$ or $t(u) = \perp$. It is easy to see that the threshold function $t(u) = \bigvee_{|S|=k} \bigwedge_{j \in S} u_j$ satisfies both properties for $k \geq \frac{n+|A|+1}{2}$. Namely, take any two vectors u, u' with $u[H] = u'[H]$, assume that there exist sets S and S' such that $u_j = x$ for all $j \in S$ and $u'_j = y$ all $j \in S'$. Then, since $|S \cap S' \cap H| \geq 2k - n - |A| > 0$, and hence $x = y$.

As usual, we consider two cases in the proof of security, depending on whether the dealer is corrupted or not.

Dishonest dealer. It is convenient to consider the case when the dealer is dishonest first, as some of the derived equations will be useful in the honest dealer case too. Beside the dealer, the players in $A \subseteq \{1, \dots, n\}$ are corrupted, and we let $H = \{1, \dots, n\} \setminus A$ be the set of honest players. We consider the real-world system (PLAYER[H] | NET') consisting of the honest participants and the network NET'. This is a system with input (s'_0, s'_A) and output (y'_H, r'_A) described by the defining

$\text{WCAST}(x', w) = (y'_1, \dots, y'_n):$ $y'_i = x' \wedge w[i] \quad (i = 1, \dots, n)$	$\text{DEALER}(x') = (s'_0):$ $s'_0[i] = x' \quad (i = 1, \dots, n)$
$\text{INT}:$ $w = \top^n$	$\text{PLAYER}[i](r'_i) = (y'_i, s'_i):$ $s'_i[j] = r'_i[0] \quad (j = 1, \dots, n)$ $y'_i = t(r'_i[1], \dots, r'_i[n])$
$\text{NET}'(s'_0, \dots, s'_n) = (r'_1, \dots, r'_n):$ $r'_i[j] = s'_j[i] \quad (i = 1, \dots, n; j = 0, \dots, n)$	

Figure 14: Weak broadcast protocol.

equations of $\text{PLAYER}[h]$ for $h \in H$ and NET' given in Figure 14. We use these equations to express each output variable of the system in terms of the input variables. For y'_h ($h \in H$) we have

$$\begin{aligned}
y'_h &= t(r'_h[1], \dots, r'_h[n]) \\
&= t(s'_1[h], \dots, s'_n[h]) \\
&= t(s'_A[h], s'_H[h]) \\
&= t(s'_A[h], r'_H[0]) = t(s'_A[h], s'_0[H]).
\end{aligned}$$

For the other output variables $r'_A[i]$ we distinguish two cases, depending on whether $i \in A$. For $a \in A$, we immediately get $r'_A[a] = s'_a[A]$. For $h \in H$, we have $r'_A[h] = s'_h[A] = (r'_h[0])^A = (s'_0[h])^A$. The resulting system is given by the following equations

$$r'_A[a] = s'_a[A] \quad (a \in A) \quad (4)$$

$$r'_A[h] = (s'_0[h])^A \quad (h \in H) \quad (5)$$

$$y'_h = t(s'_A[h], s'_0[H]) \quad (h \in H) \quad (6)$$

We now turn to the simulator. Recall that the simulator should turn the system defined by WCAST into one equivalent to the real world system. To this end, the simulator should take s'_0, s'_A and y'_A as input (from the external environment and ideal functionality respectively), and output x', w (to the ideal functionality) and r'_A (to the external environment). Notice that the simulator has all the inputs necessary to compute the values defined by the real system, and in fact can set r'_A using just those equations. The only difficulty is that the simulator cannot set y'_h directly, but has only indirect control over its value through the ideal functionality and the variables x', w . From the properties of function t , we know that all $y'_h = t(s'_A[h], s'_0[H])$ take either the same value or \perp . So, the simulator can set x' to this common value, and use w to force some y'_h to \perp as appropriate. The simulator SIM' is given in Figure 15 (right). It is easy to verify that $(\text{SIM}' \mid \text{WCAST})$ is equivalent to the real system.

Honest dealer. In this case, we first consider the real-world system $(\text{DEALER} \mid \text{PLAYER}[H] \mid \text{NET}')$ consisting of the dealer, the honest participants $H \subseteq \{1, \dots, n\}$, and the network NET' . The corrupted parties are given by the set $A = \{1, \dots, n\} \setminus H$. This is a system with input (x', s'_A) and output (y'_H, r'_A) described by the defining equations of DEALER , $\text{PLAYER}[h]$ for $h \in H$, and NET' given in Figure 14. Notice that this is a superset of the equations for the real-world system $(\text{PLAYER}[H] \mid \text{NET}')$ considered in the dishonest dealer case. So, equations (4), (5) and

(6) are still valid. Adding the equations from DEALER and using the properties of t we get that $y'_h = t(s'_A[h], s'_0[H]) = t(s'_A[h], (x)^H) = x$. Similarly, for $h \in H$, we have $r'_A[h] = (s'_0[h])^A = (x)^A$. Finally, we know from (4) that $r'_A[a] = s'_a[A]$. Combining the equations together, we get the following real system:

$$\begin{aligned} y'_h &= x' & (h \in H) \\ r'_A[a] &= s'_a[A] & (a \in A) \\ r'_A[h] &= (x')^A & (h \in H) \end{aligned}$$

We now move to the simulator. Recall that the simulator should turn the system defined by WCAST and INT into one equivalent to the real world system. To this end, the simulator should take y'_A and s'_A as input (from the ideal functionality and external environment respectively), and output r'_A . Notice that $y'_h = x'$ in the real system is defined just as in the equations for the ideal functionality WCAST when combined with the (honest) dealer interface INT. (In fact, $y'_a = x'$ also for $a \in A$.) The other variables r'_A can be easily set by the simulator as shown in Figure 15 (left). It is immediate to check that (SIM | INT | WCAST) is equivalent to the real world system.

4 Verifiable Secret Sharing

Let $\mathbb{F}_t[X]$ be the set of all polynomials of degree at most t over a finite field \mathbb{F} such that⁸ $\{0, 1, \dots, n\} \subseteq \mathbb{F}$. We consider the n -party *verifiable secret sharing* (VSS) functionality that takes as input from a *dealer* a degree- t polynomial $\mathbf{p} \in \mathbb{F}_t[X]$ and, for all $i \in [n]$, outputs the evaluation $\mathbf{p}(i)$ to the i -th party. The formal definition of VSS: $\mathbb{F}_t[X]_{\perp} \mapsto \mathbb{F}_{\perp}^n$ is given in Figure 18 (left), where by convention $\perp(x) = \perp$ for all x .

We devise a protocol implementing the VSS functionality on top of a point-to-point network functionality NET defined as in the previous section that allows the n parties to exchange elements from \mathbb{F} , and two other auxiliary functionalities. The protocol is based on the one by [5]. Even though its complexity is exponential in n , we have chosen to present this protocol due to its simplicity. The first auxiliary functionality (GRAPH) grants all parties access to the adjacency matrix of an n -vertex directed graph (with loops), where each party $i \in [n]$ can add outgoing edges to vertex i , but not to any other vertex $j \neq i$. Formally, GRAPH: $\{\perp, \top\}^n \times \dots \times \{\perp, \top\}^n \rightarrow \{\perp, \top\}^{n \times n}$ is given in Figure 18 (center). Setting $G[i, j] = \top$ is interpreted as including an edge from i to j in the graph. GRAPH can be immediately implemented using n copies of a broadcast functionality, where a different party acts as the sender in each copy. We also assume the availability of an additional

⁸This assumption is not really necessary; we could replace $\{0, 1, \dots, n\}$ with $\{0, x_1, \dots, x_n\}$ for any n distinct field elements x_1, \dots, x_n .

$$\left| \begin{array}{l} \text{SIM}(y'_A, s'_A) = (r'_A): \\ r'_A[a] = s'_a[A] \\ r'_A[h] = y'_A \end{array} \right| \begin{array}{l} (a \in A) \\ (h \in H) \end{array} \left| \begin{array}{l} \text{SIM}'(y'_A, s'_0, s'_A) = (x', w, r'_A): \\ r'_A[a] = s'_a[A] \\ r'_A[h] = (s'_0[h])^A \\ x' = \bigvee_{h \in H} t(s'_A[h], s'_0[H]) \\ w[h] = (t(s'_A[h], s'_0[H]) > \perp) \end{array} \right| \begin{array}{l} (a \in A) \\ (h \in H) \\ (h \in H) \\ (h \in H) \end{array}$$

Figure 15: Real world systems and simulators for the weak broadcast protocol. Honest dealer case (left) and dishonest dealer case (right)

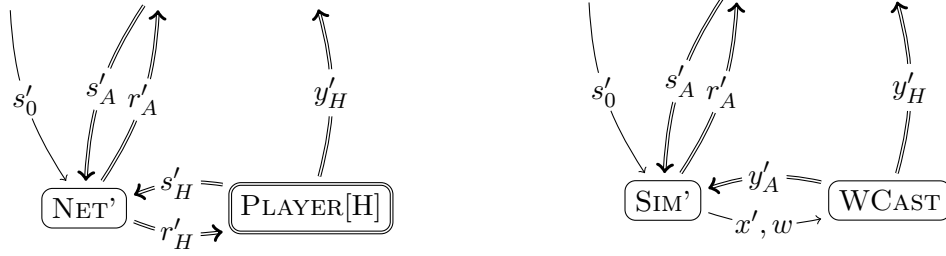


Figure 16: Security of the weak multicast protocol, when the dealer is dishonest. Real world execution on the left. Simulated attack in the ideal world on the right.

unidirectional network functionality $\text{NET}' : (\mathbb{F}_t[X]_{\perp}^2)^n \rightarrow (\mathbb{F}_t[X]_{\perp}^2)^n$ that allows the VSS dealer to send to each party a pair of polynomials of degree at most t . See Figure 18 (right).

The VSS protocol. We turn to the actual protocol securely implementing the VSS functionality. We first define some auxiliary functions. For any subset $C \subseteq [n]$, let $\text{clique}_C : \{\perp, \top\}^{n \times n} \rightarrow \{\perp, \top\}$ be the function $\text{clique}_C(G) = \bigwedge_{i,j \in C} G[i,j]$. This function is clearly monotone, and tests if C is a clique in G . For any set A , we equip the set A_{\perp} with a monotone equality-test function $\text{eq} : A_{\perp} \times A_{\perp} \rightarrow \{\perp, \top\}$ where $\text{eq}(x,y) \equiv (x = y \neq \perp)$. Monotonicity follows from the fact that all the pairs (x,x) such that $\text{eq}(x,y) = \top$ are maximal elements in $A_{\perp} \times A_{\perp}$.

For any $S \subseteq [n]$ of size $|S| \geq t+1$, and $r \in \mathbb{F}_b^n$, let $\text{interpolate}_S(r) \in \mathbb{F}_t[X]_{\perp}$ be the (unique) polynomial $h \in \mathbb{F}_t[X]$ such that $h(S) = r[S]$ if such polynomial exists, and $\text{interpolate}_S(r) = \perp$ otherwise. For $C \subseteq [n]$, define also a monotone function $\text{interpolate}_{C,t} : \mathbb{F}_{\perp}^n \rightarrow \mathbb{F}_t[Y]_{\perp}^{\top}$ where $\text{interpolate}_{C,t}(r) = \bigvee \{\text{interpolate}_S(r) : S \subseteq C, |S| = |C| - t\}$. Notice that $\text{interpolate}_{C,t}(r) = \perp$ if no interpolating polynomial exists, while $\text{interpolate}_{C,t}(r) = \top$ if there are multiple solutions. Note that if $n \geq 4t+1$ and $|C| \geq n-t$, then \top never occurs: Indeed, let $S, S' \subseteq C$ be such that $|S| = |S'| = |C| - t$, and such that both $\text{interpolate}_S(r)$ and $\text{interpolate}_{S'}(r)$ differ from \perp . Since $|S \cap S'| \geq |C| - 2t \geq n - 3t \geq t+1$, we must have $\text{interpolate}_S(r) = \text{interpolate}_{S'}(r)$ by the fact that two degree t polynomials agreeing at $t+1$ points are necessarily equal. For future reference, this is summarized by the following lemma.

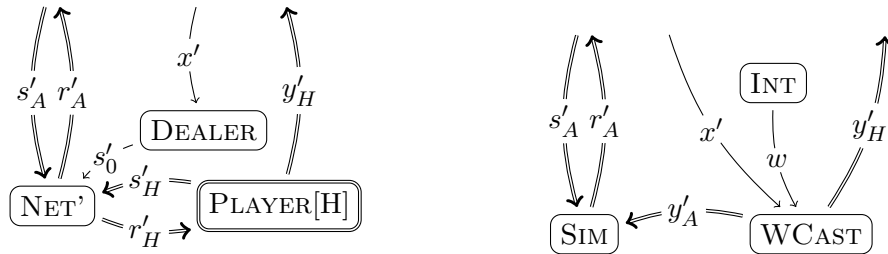


Figure 17: Security of the weak multicast protocol, when the dealer is honest. Real world execution on the left. Simulated attack in the ideal world on the right.

$$\left| \begin{array}{l} \text{VSS}(\mathbf{p}) = (p_1, \dots, p_n) \\ p_i = \mathbf{p}(i) \quad (i = 1, \dots, n) \end{array} \right| \left| \begin{array}{l} \text{GRAPH}(G_1, \dots, G_n) = G: \\ G[i, j] = G_i[j] \quad (i, j = 1, \dots, n) \end{array} \right| \left| \begin{array}{l} \text{NET}'(s') = (r'_1, \dots, r'_n): \\ r'_i = s'[i] \quad (i = 1, \dots, n) \end{array} \right| .$$

Figure 18: The VSS functionality, and two auxiliary functions used to realize it.

$$\left| \begin{array}{l} \text{DEALER}(\mathbf{p}) = s': \\ \mathbf{f} \leftarrow \mathcal{P}_t(\mathbf{p}) \\ s'[i] = (\mathbf{f}(\cdot, i), \mathbf{f}(i, \cdot)) \quad (i = 1, \dots, n) \end{array} \right| \left| \begin{array}{l} \text{PLAYER}[i](r'_i, G, r_i) = (p_i, s_i, G_i): \quad (i = 1, \dots, n) \\ (\mathbf{g}_i, \mathbf{h}_i) = r'_i \\ s_i[j] = \mathbf{g}_i(j) \quad (j = 1, \dots, n) \\ G_i[j] = \text{eq}(r_i[j], \mathbf{h}_i(j)) \quad (j = 1, \dots, n) \\ \mathbf{o}_i = \bigvee_{\substack{C \subseteq [n] \\ |C| \geq n-t}} [\text{clique}_C(G) \wedge \text{interpolate}_{C,t}(r_i)] \\ p_i = \mathbf{o}_i(0) \end{array} \right| .$$

Figure 19: The VSS protocol.

Lemma 1 *Let n, t be such that $n \geq 4t + 1$, and let $C \subseteq [n]$ be such that $|C| \geq n - t$. Then $\text{interpolate}_{C,t}(r) \neq \top$ for all $r \in \mathbb{F}_{\perp}^n$.*

In the following, denote as $\mathbb{F}_t[X, Y]$ the set of polynomials $\mathbf{f} = \mathbf{f}(X, Y)$ in $\mathbb{F}[X, Y]$ with degree at most t in each variable X and Y . For any $\mathbf{p} \in \mathbb{F}_t[X]_{\perp}$, let $\mathcal{P}_t(\mathbf{p})$ the (uniform distribution over the) set of bivariate polynomials $\mathbf{f} = \mathbf{f}(X, Y) \in \mathbb{F}_t[X, Y]_{\perp}$ of degree at most t in X and Y such that $\mathbf{f}(\cdot, 0) = \mathbf{p}$. (By convention, if $\mathbf{p} = \perp$, then $\mathcal{P}_t(\mathbf{p}) = \{\perp\}$.)

The protocol consists of a dealer **DEALER** which, on input a polynomial \mathbf{p} , first chooses a random bivariate polynomial \mathbf{f} in $\mathcal{P}_t(\mathbf{p})$. (This is the only random choice of the entire protocol.) For all $i \in [n]$, it sends the two polynomials $\mathbf{g}_i = \mathbf{f}(\cdot, i)$ and $\mathbf{h}_i = \mathbf{f}(i, \cdot)$ to player i , with the usual convention that if $\mathbf{f} = \perp$, then $\mathbf{f}(\cdot, i) = \mathbf{f}(i, \cdot) = \perp$. The players then determine whether the polynomials they received are consistent. This is achieved by having each honest party i send $\mathbf{g}_i(j)$ to player j , who, in turn, checks consistency with $\mathbf{h}_j(i)$. (Note that if the polynomials are correct, then $\mathbf{g}_i(j) = \mathbf{h}_j(i) = \mathbf{f}(j, i)$.) If the consistency check is successful, player j raises the entry $G[j, i]$ to \top . Each honest party i then waits for a clique $C \subseteq [n]$ of size (at least) $n - t$ to appear in the directed graph defined by G , and computes the polynomial $\mathbf{o}_i \in \mathbb{F}_t[X]_{\perp}^{\top}$ obtained interpolating the values $\mathbf{g}_j(i)$ received from other parties. (Here \top represents an error condition meaning that multiple interpolating polynomials were found, and should not really occur in actual executions, as we will show.) As soon as such a polynomial is found, the honest party terminates with output $p_i = \mathbf{o}_i(0)$. A formal specification is given in Figure 19.

In the following, we turn to proving security of the protocol. The analysis consists of two cases.

Honest dealer security. We start by analyzing the security of the above protocol in the case where the dealer is honest. For all $A \subseteq [n]$ where $|A| = t$ and $n \geq 4t + 1$, define $H = [n] \setminus A$. When the players in the set A are corrupted (and thus the players in H are honest), an execution of the VSS protocol with honest dealer is given by the system $(\text{DEALER} \mid \text{PLAYER}[H] \mid \text{NET}' \mid \text{NET} \mid \text{GRAPH})$ with inputs \mathbf{p}, s_A, G_A and outputs $r_D[A], r_A, G, p_H$ given in Figure 20.

We proceed by combining all the equations together, and simplifying the result, until we obtain a system of equations that can be easily simulated. We use the above definition of the system to

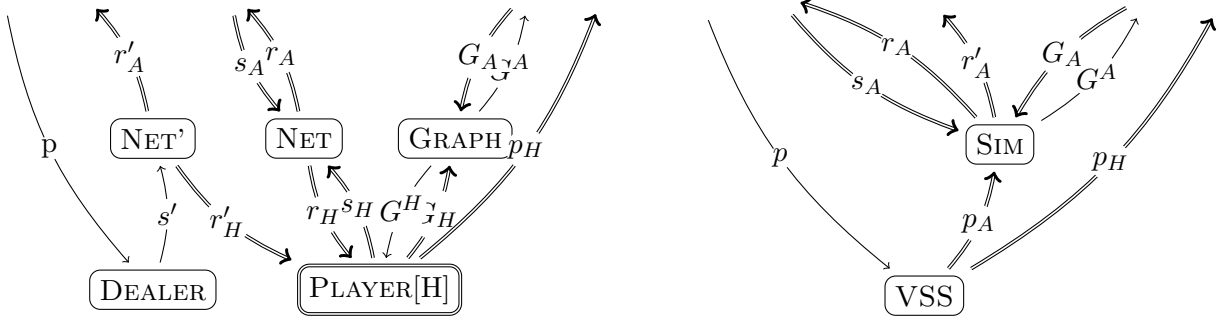


Figure 20: Security of the VSS protocol when the dealer is honest.

obtain the following equations describing $(\text{DEALER} \mid \text{PLAYER}[H] \mid \text{NET}' \mid \text{NET} \mid \text{GRAPH})$: For any $i, j \in [n]$, and any $h \in H, a \in A$, we have

$$\begin{aligned}
 \mathbf{f} &\stackrel{\$}{\leftarrow} \mathcal{P}_t(\mathbf{p}) & G[h, j] &= \text{eq}(r_h[j], \mathbf{f}(h, j)) \\
 r'_i &= (\mathbf{f}(\cdot, i), \mathbf{f}(i, \cdot)) & G[a, j] &= G_a[j] \\
 r_i[h] &= \mathbf{f}(i, h) & \mathbf{o}_h &= \bigvee_{C \subseteq [n], |C| \geq n-t} [\text{clique}_C(G) \wedge \text{interpolate}_{C,t}(r_h)] \\
 r_i[a] &= s_a[i] & p_h &= \mathbf{o}_h(0) .
 \end{aligned}$$

For convenience, some simplifications have already been made: First \mathbf{g}_i and \mathbf{h}_i have been replaced by $\mathbf{f}(\cdot, i)$ and $\mathbf{f}(i, \cdot)$, respectively. Second, we used the facts that $r'_i = s'_i[i]$ and $r_i[h] = s_h[i] = \mathbf{f}(i, h)$ for all $h \in H$ and all $i \in [n]$ by the definitions of the network functionalities NET' and NET . Finally, we have set values for $G[\cdot, \cdot]$ according to the protocol specification (for honest players) and the inputs G_a of players $a \in A$.

In order to further simplify the system, we claim that $p_h = \mathbf{p}(h)$ for $h \in H$. If $\mathbf{p} = \perp$, then this is easy to see because $\mathbf{f} = \perp$ and $G[h, j] = \text{eq}(r_h[j], \perp) = \perp$. Therefore, we necessarily have $\text{clique}_C(G) = \perp$ for all $C \subseteq [n]$ with $|C| \geq n - t$, since $|C \cap H| \geq n - 2t > 0$. So, we only need to prove the claim for $\mathbf{p} \neq \perp$. Notice that the equations $G[h, j] = \text{eq}(r_h[j], \mathbf{f}(h, j))$, depending on whether $j = h' \in H$ or $j = a \in A$, can be replaced by the set of equations

$$\begin{aligned}
 G[h, h'] &= \text{eq}(r_h[h'], \mathbf{f}(h, h')) = \text{eq}(\mathbf{f}(h, h'), \mathbf{f}(h, h')) = \top \\
 G[h, a] &= \text{eq}(r_h[a], \mathbf{f}(h, a)) = \text{eq}(s_a[h], \mathbf{f}(h, a)) .
 \end{aligned}$$

This in particular implies that $C = H$ is a clique of size at least $n - t$ in the graph defined by G , i.e., we have $\text{clique}_H(G) = \top$ by the above. Also, since $r_h[h'] = \mathbf{f}(h, h')$, we necessarily have

$$\mathbf{o}_h \geq \text{clique}_H(G) \wedge \text{interpolate}_{H,t}(r_h) = \top \wedge \mathbf{f}(h, \cdot) = \mathbf{f}(h, \cdot)$$

by Lemma 1. Now, let $S \subseteq C$ be any sets such that $|C| \geq n - t$ and $|S| = |C| - t \geq n - 2t$. Since $\mathbf{o}_h(h') = \mathbf{f}(h, h')$ for all $h' \in H$ and $|S \cap H| \geq n - 3t \geq t + 1$, we have $\text{interpolate}_S(r_h) \geq \mathbf{f}(h, \cdot)$, and, by Lemma 1, $\text{interpolate}_S(r_h) = \mathbf{f}(h, \cdot)$. This proves that $\mathbf{o}_h = \text{interpolate}_{C,t}(r_h) = \mathbf{f}(h, \cdot)$, and $p_h = \mathbf{o}_h(0) = \mathbf{f}(h, 0) = \mathbf{p}(h)$.

Summarizing, the real system is described by the following set of equations:

$$\begin{array}{ll}
f \stackrel{\$}{\leftarrow} \mathcal{P}_t(\mathbf{p}) & G[h, h'] = (\mathbf{p} > \perp) \\
r'_a = (f(\cdot, a), f(a, \cdot)) & G[h, a] = \text{eq}(s_a[h], f(h, a)) \\
r_a[a'] = s_{a'}[a] & G[a, j] = G_a[j] \\
r_a[h] = f(a, h) & p_h = \mathbf{p}(h) .
\end{array}$$

Notice that this is exactly how p_h is defined by the VSS functionality. So, in order to prove security, it is enough to give a simulator SIM that on input p_A, s_A, G_A , outputs G, r_A and r'_A as defined in the above system of equations. See Figure 20 (right).

The problem faced by the simulator is that it cannot test $\mathbf{p} > \perp$ and generate f as in the equations because it does not know the value of \mathbf{p} , rather it only has partial information $p_A = \mathbf{p}(A)$. The first condition $\mathbf{p} > \perp$ is easy to check because it is equivalent to $p_a = \mathbf{p}(a) > \perp$ for any $a \in A$. In order to complete the simulation, we observe that the equations only depend on the $2t$ polynomials $f(\cdot, A)$ and $f(A, \cdot)$. The next lemma shows that, given $\mathbf{p}(A)$, the polynomials $f(\cdot, A)$ and $f(A, \cdot)$ are statistically independent from \mathbf{p} , and their distribution can be easily sampled.

Lemma 2 *Let $\mathbf{p} \in \mathbb{F}_t[X]$, let $f \stackrel{\$}{\leftarrow} \mathcal{P}_t(\mathbf{p})$, and for all $a \in A$, let $\mathbf{g}_a = f(\cdot, a)$ and $\mathbf{h}_a = f(a, \cdot)$. The conditional distribution of $(\mathbf{g}_a, \mathbf{h}_a)_{a \in A}$ given $\mathbf{p}(A)$ is statistically independent of \mathbf{p} , and it can be generated by the following algorithm $\text{Samp}(p_A)$: first pick random polynomials $\mathbf{h}_a \in \mathbb{F}_t[Y]$ independently and uniformly at random subject to the constraint $\mathbf{h}_a(0) = p_a$. Then, pick $\mathbf{g}_a \in \mathbb{F}_t[X]$, also independently and uniformly at random, subject to the constraint $\mathbf{g}_a(A) = \mathbf{h}_A(a)$.*

Using the algorithm from the lemma, we obtain the following simulator SIM :

$$\left| \begin{array}{l}
\text{SIM}(p_A, s_A, G_A) = (G^A, r_A, r'_A): \\
(\mathbf{g}_A, \mathbf{h}_A) \leftarrow \text{Samp}(p_A) \\
r'_A = (\mathbf{g}_A, \mathbf{h}_A) \\
r_a[h] = \mathbf{h}_a(h) \quad (h \in H, a \in A) \\
r_a[a'] = s_{a'}[a] \quad (a, a' \in A)
\end{array} \right| \quad \left| \begin{array}{l}
G[h, h'] = \bigvee_{a \in A} (p_a > \perp) \quad (h, h' \in H) \\
G[h, a] = \text{eq}(s_a[h], \mathbf{g}_a(h)) \quad (h \in H, a \in A) \\
G[a, j] = G_a[j] \quad (a \in A, j \in [n])
\end{array} \right|$$

As usual, if $\mathbf{p} = \perp$, then $p_A = \perp^A$ and by convention $\text{Samp}(p_A) = \{\perp, \perp\}$.

Dishonest dealer security. We now look at the case where the dealer is not honest. As above, for all $A \subseteq [n]$ where $|A| = t$ and $n \geq 4t + 1$, define $H = [n] \setminus A$. When the players in the set A are corrupted (and thus the players in H are honest), an execution of the VSS protocol with dishonest dealer is given by the system ($\text{PLAYER}[H] \mid \text{NET}' \mid \text{NET} \mid \text{GRAPH}$) with inputs s', r_A, G_A , and outputs r'_A, s_A, p_H and G^A . As above, we start with an equational description of the system, and will simplify it below into a form where the construction of a corresponding simulator becomes obvious. For all $i, j \in [n]$, $h, h' \in H$, and $a \in A$, we have

$$\begin{array}{ll}
(\mathbf{g}_h, \mathbf{h}_h) = s'[h] & G[h, a] = \text{eq}(s_a[h], \mathbf{h}_h(a)) \\
r'_a = s'[a] & G[a, j] = G_a[j] \\
r_i[h] = \mathbf{g}_h(i) & \mathbf{o}_h = \bigvee_{C \subseteq [n], |C| \geq n-t} [\text{clique}_C(G) \wedge \text{interpolate}_{C,t}(r_h)] \\
r_i[a] = s_a[i] & p_h = \mathbf{o}_h(0) . \\
G[h, h'] = \text{eq}(\mathbf{g}_{h'}(h), \mathbf{h}_h(h')) &
\end{array}$$

Notice that we have already undertaken several easy simplification steps, defining variables which are part of the output as a function of the system inputs and of auxiliary variables \mathbf{g}_H , \mathbf{h}_H , \mathbf{o}_H , and r_H . Specifically, to obtain the above equations starting from the original system specification, we have used $r_i[j] = s_j[i]$, where $s_h[i] = \mathbf{g}_h(i)$, together with $r'_i = s'[i]$ and the definition of $G[h, i]$, distinguishing between the cases $i = a \in A$ and $i = h' \in H$.

Recall that in order to prove security, we need to give a simulator SIM with input s', G_A, s_A, p_A and output r'_A, r_A, G^A and \mathbf{p} such that $(\text{VSS} \mid \text{SIM})$ is equivalent to the above system. (See Figure 21.) Notice that in the system describing a real execution, all variables except p_h (and intermediate value \mathbf{o}_h) are defined as functions of the inputs given to the simulator. So, SIM can set all these variables just as in the system describing the real execution. The only difference between a real execution and a simulation is that the simulator is not allowed to set p_h directly. Rather, it should specify a polynomial $\mathbf{p} \in \mathbb{F}_t[X]_\perp$, which implicitly defines $p_h = \mathbf{p}(h)$ through the equations of the ideal VSS functionality. In other words, in order to complete the description of the simulator we need to show that SIM can determine such a polynomial \mathbf{p} based on its inputs s', G_A, s_A, p_A such that $\mathbf{p}(h)$ equals p_h as defined by the above system of equations.

Before defining \mathbf{p} , we recall the following lemma whose simple proof is standard and omitted:

Lemma 3 *Let S be such that $|S| \geq t+1$ and let $\{\mathbf{g}_h, \mathbf{h}_h\}_{h \in S}$ be a set of $2 \cdot |S|$ polynomials of degree t . Then, $\mathbf{g}_h(h') = \mathbf{h}_{h'}(h)$ for all $h, h' \in S$ holds if and only if there exists a unique polynomial $\mathbf{f} \in \mathbb{F}_t[X, Y]$ such that $\mathbf{f}(\cdot, h) = \mathbf{g}_h$ and $\mathbf{f}(h, \cdot) = \mathbf{h}_h$ for all $h \in S$.*

For $T \subseteq H$, $|T| \geq t+1$, define $\text{interpolate2}_T(s')$ to be the (unique) polynomial $\mathbf{f} \in \mathbb{F}_t[X, Y]$ such that $\mathbf{f}(\cdot, h) = \mathbf{g}_h$ and $\mathbf{f}(h, \cdot) = \mathbf{h}_h$ for all $h \in T$ (if it exists), and \perp otherwise or if $s'[h] = \perp$ for some $h \in T$. Also, given $C \subseteq [n]$, define

$$\text{interpolate2}_C(s') = \bigvee \{ \text{interpolate2}_S(s') : S \subseteq C, |S| \geq |C| - t \}.$$

Note that since $|C| \geq n - t$ and $n \geq 4t + 1$, $\text{interpolate2}_C(s') \neq \top$. Indeed, for any two $S, S' \subseteq C$ such that both $\text{interpolate2}_S(s')$ and $\text{interpolate2}_{S'}(s')$ differ from \perp , we have $|S \cap S'| \geq t+1$ and hence $\text{interpolate2}_S(s') = \text{interpolate2}_{S \cap S'}(s') = \text{interpolate2}_{S'}(s')$ by Lemma 3. We finally define the polynomial $\mathbf{p} = \mathbf{f}(\cdot, 0)$, where

$$\tilde{\mathbf{f}} = \bigvee_{C \subseteq [n], |C| \geq n-t} \text{clique}_C(G) \wedge \text{interpolate2}_C(s'). \quad (7)$$

We first prove that $\mathbf{p} < \top$: To this end, assume that $\mathbf{p} \neq \perp$. Then, $\tilde{\mathbf{f}} \neq \perp$, and there must exist $C \subseteq [n]$ such that $\text{clique}_C(G) = \top$. Let $S = C \cap H$. Note that for all $h, h' \in S$, since $G[h, h'] = \top$, it must be that $\mathbf{h}_h(h') = \mathbf{g}_{h'}(h)$. Therefore, since $|S| \geq n - 2t > 2t + 1$, by Lemma 3, there exists a unique polynomial \mathbf{f}_C such that $\mathbf{f}(\cdot, h) = \mathbf{g}_h$ and $\mathbf{f}(h, \cdot) = \mathbf{h}_h$ for all $h \in S$, and by the above

$$\mathbf{f}_C = \text{interpolate2}_C(s') = \text{interpolate2}_{C \cap H}(s').$$

Now assume that there exist two such cliques C and C' , with $S = C \cap H$ and $S' = C' \cap H$. Then, since

$$|S \cap S'| = |S| + |S'| - |S \cup S'| \geq 2(|C| - |A|) - |H| \geq n - 3|A| \geq t + 1, \quad (8)$$

by Lemma 3, we necessarily have $\mathbf{f}_C = \mathbf{f}_{C'} = \tilde{\mathbf{f}}$.

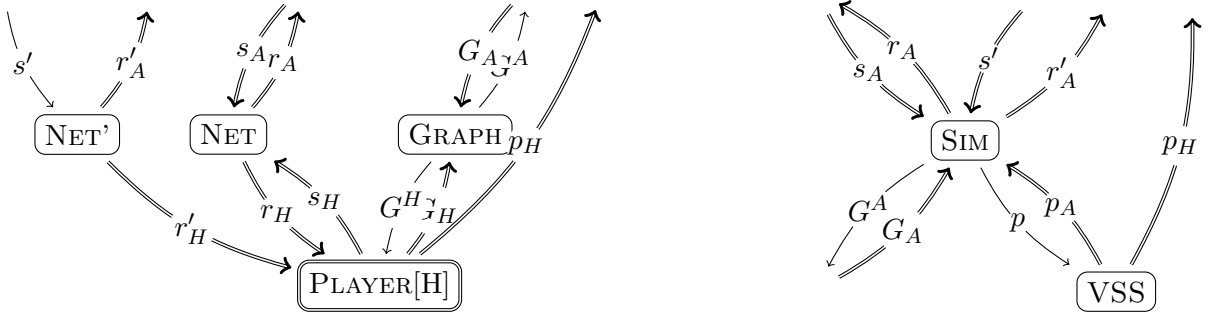


Figure 21: Security of the VSS protocol when the dealer is dishonest.

Finally, it is easy to see that $p(h) = p_h$ for all $h \in H$. Namely, if there exists $C \subseteq [n]$ with $\text{clique}_C(G) = \top$, then $r_h[h'] = g_{h'}(h) = \tilde{f}(h, h')$ for all $h' \in C \cap H$, and therefore $\mathbf{o}_h = \text{interpolate}_C(r_h) = \text{interpolate}_{C \cap H}(r_h) = f(h, \cdot)$, and thus $p_h = \mathbf{o}_h(0) = \tilde{f}(h, 0) = p(h)$.

We therefore conclude that the real system is equivalent to $(\text{SIM} \mid \text{VSS})$ where SIM is the simulator defined by the following equations:

$\begin{aligned} \text{SIM}(s', G_A, s_A, p_A) &= (r'_A, r_A, G^A, \mathbf{p}): \\ r'_a &= s'[a] & (a \in A) \\ r_a[h] &= g_h(a) & (h \in H, a \in A) \\ r_a[a'] &= s_{a'}[a] & (a, a' \in A) \\ G[h, h'] &= \text{eq}(g_{h'}(h), h_h(h')) & (h, h' \in H) \end{aligned}$	$\begin{aligned} G[h, a] &= \text{eq}(s_a[h], h_h(a)) & (a \in A, h \in H) \\ G[a, j] &= G_a[j] & (a \in A, j \in [n]) \\ \tilde{f} &= \bigvee_{\substack{C \subseteq [n] \\ C \geq n-t}} \text{clique}_C(G) \wedge \text{interpolate2}_C(s') \\ \mathbf{p} &= \tilde{f}(\cdot, 0) \end{aligned}$
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

5 Conclusions

Recognizing the inherent hardness of delivering security proofs for complex cryptographic protocols that are both precise and intuitive within existing security frameworks, we have presented a new framework to study the security of multi-party computations based on equational descriptions of interactive processes. Our framework allows a simple and intuitive, yet completely formal, description of interactive processes via sets of equations, and its foundations rely on tools from programming-language theory and domain theory. Beyond its simplicity, our framework completely avoids explicit addressing of non-determinism within cryptographic security proofs, making security proofs a matter of simple equational manipulations over precise mathematical structures. As a case study, we have presented simple security analyses of (variants of) two classical asynchronous protocols within our framework, Bracha's broadcast protocol [8] and the Ben-Or, Canetti, Goldreich VSS protocol [5].

We are convinced that our work will open up the avenue to several directions for future work. First off, while the results in this paper are presented for the special case of perfect security, a natural next step is to extend the framework to statistical and even computational security. Moreover, while the expressiveness of our framework (i.e., the monotonicity restrictions on protocols) remains to be thoroughly investigated, most distributed protocols we examined so far, seemed to admit a representation within our framework, possibly after minor modifications which often resulted in a

simpler protocol description. For this reason, our thesis is that this holds true for *all* protocols of interest, and that non-monotonicity, as a source of unnecessary complexity and proof mistakes, should be avoided whenever possible.

References

- [1] S. Abramsky and A. Jung. *Handbook of Logic in Computer Science*, volume III, chapter Domain theory, pages 1–168. Oxford University Press, 1994.
- [2] Michael Backes, Birgit Pfitzmann, and Michael Waidner. A general composition theorem for secure reactive systems. In *TCC 2004*, volume 2951 of *Lecture Notes in Computer Science*, pages 336–354, 2004.
- [3] Donald Beaver and Shafi Goldwasser. Multiparty computation with faulty majority. In *CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*, pages 589–590, 1989.
- [4] Assaf Ben-David, Noam Nisan, and Benny Pinkas. FairplayMP: a system for secure multiparty computation. In *ACM Conference on Computer and Communications Security*, pages 257–266, 2008.
- [5] Michael Ben-Or, Ran Canetti, and Oded Goldreich. Asynchronous secure computation. In *STOC*, pages 52–61, 1993.
- [6] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC '88*, pages 1–10, 1988.
- [7] Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas P. Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael I. Schwartzbach, and Tomas Toft. Secure multiparty computation goes live. In *Financial Cryptography (FC 2009)*, volume 5628 of *Lecture Notes in Computer Science*, pages 325–343, 2009.
- [8] Gabriel Bracha. An asynchronous $[(n-1)/3]$ -resilient consensus protocol. In *PODC '84*, pages 154–162, 1984.
- [9] Ran Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptology*, 13(1):143–202, 2000.
- [10] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS '01*, pages 136–145, 2001.
- [11] Ivan Damgård, Marcel Keller, E. Larraia, C. Miles, and Nigel P. Smart. Implementing aes via an actively/covertly secure dishonest-majority mpc protocol. *IACR Cryptology ePrint Archive*, 2012.
- [12] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC '87*, pages 218–229, 1987.

- [13] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.
- [14] Carl A. Gunter. *Semantics of programming languages - structures and techniques*. Foundations of computing. MIT Press, 1993.
- [15] Dennis Hofheinz and Victor Shoup. Gnuc: A new universal composability framework. *IACR Cryptology ePrint Archive*, 2011.
- [16] G. Kahn. The semantics of a simple language for parallel programming. In J. L. Rosenfeld, editor, *Information processing*, pages 471–475, Stockholm, Sweden, Aug 1974. North Holland, Amsterdam.
- [17] Vladimir Kolesnikov and Thomas Schneider. A practical universal circuit construction and secure evaluation of private functions. In *Financial Cryptography (FS 2008)*, volume 5143 of *Lecture Notes in Computer Science*, pages 83–97, 2008.
- [18] Ralf Küsters. Simulation-based security with inexhaustible interactive turing machines. In *19th IEEE Computer Security Foundations Workshop, (CSFW-19 2006)*, pages 309–320, 2006.
- [19] Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella. Fairplay - secure two-party computation system. In *USENIX Security Symposium*, pages 287–302, 2004.
- [20] Ueli Maurer and Renato Renner. Abstract cryptography. In *Innovations in Computer Science - ICS 2010*, pages 1–21, 2011.
- [21] Ueli M. Maurer. Indistinguishability of random systems. In *EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 110–132, 2002.
- [22] Silvio Micali and Phillip Rogaway. Secure computation (abstract). In *CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 392–404, 1991.
- [23] Birgit Pfitzmann and Michael Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *IEEE Symposium on Security and Privacy*, pages 184–, 2001.
- [24] Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. Secure two-party computation is practical. In *ASIACRYPT 2009*, volume 5912 of *Lecture Notes in Computer Science*, pages 250–267, 2009.
- [25] N. Saheb-Djahromi. Cpo’s of measures for nondeterminism. *Theor. Comput. Sci.*, 12:19–37, 1980.
- [26] D. A. Schmidt. *Denotational Semantics*. Allyn and Bacon, 1986.
- [27] J. E. Stoy. *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory*. MIT Press, 1977.
- [28] Glynn Winskel. *The formal semantics of programming languages - an introduction*. Foundation of computing series. MIT Press, 1993.

- [29] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *FOCS '82*, pages 160–164, 1982.

Semantic Security for the Wiretap Channel

Mihir Bellare¹, Stefano Tessaro², and Alexander Vardy³

¹ Department of Computer Science & Engineering, University of California San Diego, cseweb.ucsd.edu/~mihir/

² CSAIL, Massachusetts Institute of Technology, people.csail.mit.edu/tessaro/

³ Department of Electrical & Computer Engineering, University of California San Diego, <http://www.ece.ucsd.edu/~avardy/>

Abstract. The wiretap channel is a setting where one aims to provide information-theoretic privacy of communicated data based solely on the assumption that the channel from sender to adversary is “noisier” than the channel from sender to receiver. It has developed in the Information and Coding (I&C) community over the last 30 years largely divorced from the parallel development of modern cryptography. This paper aims to bridge the gap with a cryptographic treatment involving advances on two fronts, namely definitions and schemes. On the first front (definitions), we explain that the *mis-r* definition in current use is weak and propose two alternatives: *mis* (based on mutual information) and *ss* (based on the classical notion of semantic security). We prove them equivalent, thereby connecting two fundamentally different ways of defining privacy and providing a new, strong and well-founded target for constructions. On the second front (schemes), we provide the first explicit scheme with all the following characteristics: it is proven to achieve both security (*ss* and *mis*, not just *mis-r*) and decodability; it has optimal rate; and both the encryption and decryption algorithms are proven to be polynomial-time.

1 Introduction

The wiretap channel is a setting where one aims to obtain information-theoretic privacy under the sole assumption that the channel from sender to adversary is “noisier” than the channel from sender to receiver. Introduced by Wyner, Csiszár and Körner in the late seventies [41, 14], it has developed in the Information and Coding (I&C) community largely divorced from the parallel development of modern cryptography. This paper aims to bridge the gap with a cryptographic treatment involving advances on two fronts.

The first is *definitions*. We explain that the security definition in current use, that we call *mis-r* (mutual-information security for random messages) is weak and insufficient to provide security of applications. We suggest strong, new definitions. One, that we call *mis* (mutual-information security), is an extension of *mis-r* and thus rooted in the I&C tradition and intuition. Another, semantic security (*ss*), adapts the cryptographic “gold standard” emanating from [19] and

universally accepted in the cryptographic community. We prove the two equivalent, thereby connecting two fundamentally different ways of defining privacy and providing a new, strong and well-founded target for constructions.

The second is *schemes*. Classically, the focus of the I&C community has been proofs of existence of mis-r schemes of optimal rate. The proofs are non-constructive so that the schemes may not even be explicit let alone polynomial time. Recently, there has been progress towards explicit mis-r schemes [30, 29, 21]. We take this effort to its full culmination by providing the first explicit construction of a scheme with all the following characteristics: it is proven to achieve security (not just mis-r but ss and mis) over the adversary channel; it is proven to achieve decodability over the receiver channel; it has optimal rate; and both the encryption and decryption algorithms are proven to be polynomial-time.

Today the possibility of realizing the wiretap setting in wireless networks is receiving practical attention and fueling a fresh look at this area. Our work helps guide the choice of security goals and schemes. Let us now look at all this in more detail.

THE WIRETAP MODEL. The setting is depicted in Fig. 1. The sender applies to her message M a randomized encryption algorithm $\mathcal{E}: \{0,1\}^m \rightarrow \{0,1\}^c$ to get what we call the *sender ciphertext* $X \leftarrow^* \mathcal{E}(M)$. This is transmitted to the receiver over the receiver channel ChR so that the latter gets a *receiver ciphertext* $Y \leftarrow^* \text{ChR}(X)$ which he decrypts via algorithm \mathcal{D} to recover the message. The adversary’s wiretap is modeled as another channel ChA and she accordingly gets an *adversary ciphertext* $Z \leftarrow^* \text{ChA}(X)$ from which she tries to glean whatever she can about the message.

A *channel* is a randomized function specified by a transition probability matrix W where $W[x,y]$ is the probability that input x results in output y . Here x,y are strings. The canonical example is the Binary Symmetric Channel BSC_p with crossover probability $p \leq 1/2$ taking a binary string x of any length and returning the string y of the same length formed by flipping each bit of x independently with probability p . For concreteness and simplicity of exposition we will often phrase discussions in the setting where ChR, ChA are BSCs with crossover probabilities $p_R, p_A \leq 1/2$ respectively, but our results apply in much greater generality. In this case the assumption that ChA is “noisier” than ChR corresponds to the assumption that $p_R < p_A$. This is the only assumption made: the adversary is computationally unbounded, and the scheme is keyless, meaning sender and receiver are not assumed to a priori share any information not known to the adversary.

The setting now has a literature encompassing hundreds of papers. (See the survey [28] or the book [6].) Schemes must satisfy two conditions, namely *decoding* and *security*. The *decoding* condition asks that the scheme provide error-correction over the receiver channel, namely the limit as $k \rightarrow \infty$ of the maximum, over all M of length m , of $\Pr[\mathcal{D}(\text{ChR}(\mathcal{E}(M))) \neq M]$, is 0, where k is an underlying security parameter of which m, c are functions. The original security condition of [41] was that $\lim_{k \rightarrow \infty} \mathbf{I}(M; \text{ChA}(\mathcal{E}(M)))/m = 0$ where the message random vari-

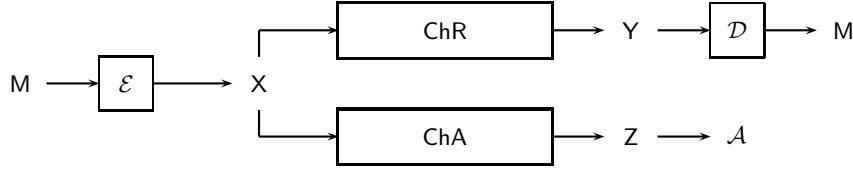


Fig. 1. Wiretap channel model. See text for explanations.

able M is uniformly distributed over $\{0, 1\}^m$ and $\mathbf{I}(M; Z) = \mathbf{H}(M) - \mathbf{H}(M|Z)$ is the mutual information. This was critiqued by [31, 32] who put forth the stronger security condition now in use, namely that $\lim_{k \rightarrow \infty} \mathbf{I}(M; \text{ChA}(\mathcal{E}(M))) = 0$, the random variable M continuing to be uniformly distributed over $\{0, 1\}^m$. The *rate* $\mathbf{Rate}(\mathcal{E})$ of a scheme is m/c .

The literature has focused on determining the maximum possible rate. (We stress that the maximum is over all possible schemes, not just ones that are explicitly given or polynomial time.) Shannon’s seminal result [37] says that if we ignore security and merely consider achieving the decoding condition then this optimal rate is the receiver channel capacity, which in the BSC case is $1 - h_2(p_R)$ where h_2 is the binary entropy function $h_2(p) = -p \lg(p) - (1 - p) \lg(1 - p)$. He gave non-constructive proofs of existence of schemes meeting capacity. Coming in with this background and the added security condition, it was natural for the wiretap community to follow Shannon’s lead and begin by asking what is the maximum achievable rate subject to both the security and decoding conditions. This optimal rate is called the secrecy capacity and, in the case of BSCs, equals the difference $(1 - h_2(p_R)) - (1 - h_2(p_A)) = h_2(p_A) - h_2(p_R)$ in capacities of the receiver and adversary channels. Non-constructive proofs of the existence of schemes with this optimal rate were given in [41, 14, 7]. Efforts to obtain explicit, secure schemes of optimal rate followed [40, 33, 30, 29, 21].

CONTEXT. Practical interest in the wiretap setting is escalating. Its proponents note two striking benefits over conventional cryptography: (1) no computational assumptions, and (2) no keys and hence no key distribution. Item (1) is attractive to governments who are concerned with long-term security and worried about quantum computing. Item (2) is attractive in a world where vulnerable, low-power devices are proliferating and key-distribution and key-management are unsurmountable obstacles to security. The practical challenge is to realize a secrecy capacity, meaning ensure by physical means that the adversary channel is noisier than the receiver one. The degradation with distance of radio communication signal quality is the basis of several approaches being investigated for wireless settings. Government-sponsored Ziva Corporation [42] is using optical techniques to build a receiver channel in such a way that wiretapping results in a degraded channel. A program called Physical Layer Security aimed at practical realization of the wiretap channel is the subject of books [6] and conferences [24]. All this activity means that schemes are being sought for implementation. If so, we need privacy definitions that yield security in applications, and we need con-

structive results yielding practical schemes achieving privacy under these definitions. This is what we aim to supply.

DEFINITIONS. A security *metric* \mathbf{xs} associates to encryption function $\mathcal{E}: \{0, 1\}^m \rightarrow \{0, 1\}^c$ and adversary channel ChA a number $\mathbf{Adv}^{\mathbf{xs}}(\mathcal{E}; \text{ChA})$ that measures the maximum “advantage” of an adversary in breaking the scheme under metric \mathbf{xs} . For example, the metric underlying the current, above-mentioned security condition is $\mathbf{Adv}^{\text{mis-r}}(\mathcal{E}; \text{ChA}) = \mathbf{I}(\mathbf{M}; \text{ChA}(\mathcal{E}(\mathbf{M})))$ where \mathbf{M} is uniformly distributed over $\{0, 1\}^m$. We call this the *mis-r* (mutual-information security for random messages) metric because messages are assumed to be random. From the cryptographic perspective, this is extraordinarily weak, for we know that real messages are not random. (They may be files, votes or any type of structured data, often with low entropy. Contrary to a view in the I&C community, compression does *not* render data random, as can be seen from the case of votes, where the message space has very low entropy.) This leads us to suggest a stronger metric that we call *mutual-information security*, defined via $\mathbf{Adv}^{\text{mis}}(\mathcal{E}; \text{ChA}) = \max_{\mathbf{M}} \mathbf{I}(\mathbf{M}; \text{ChA}(\mathcal{E}(\mathbf{M})))$ where the maximum is over all random variables \mathbf{M} over $\{0, 1\}^m$, regardless of their distribution.

At this point, we have a legitimate metric, but how does it capture privacy? The intuition is that it is measuring the difference in the number of bits required to encode the message before and after seeing the ciphertext. This intuition is alien to cryptographers, whose metrics are based on much more direct and usage-driven privacy requirements. Cryptographers understand since [19] that encryption must hide all partial information about the message, meaning the adversary should have little advantage in computing a function of the message given the ciphertext. (Examples of partial information about a message include its first bit or even the XOR of the first and second bits.) The *mis-r* and *mis* metrics ask for nothing like this and are based on entirely different intuition. We extend Goldwasser and Micali’s semantic security [19] definition to the wiretap setting, defining $\mathbf{Adv}^{\text{ss}}(\mathcal{E}; \text{ChA})$ as

$$\max_{f, \mathbf{M}} \left(\max_{\mathcal{A}} \Pr[\mathcal{A}(\text{ChA}(\mathcal{E}(\mathbf{M}))) = f(\mathbf{M})] - \max_{\mathcal{S}} \Pr[\mathcal{S}(m) = f(\mathbf{M})] \right).$$

Within the parentheses is the maximum probability that an adversary \mathcal{A} , given the adversary ciphertext, can compute the result of function f on the message, minus the maximum probability that a simulator \mathcal{S} can do the same given only the length of the message. We also define a distinguishing security (ds) metric as an analog of indistinguishability [19] via

$$\mathbf{Adv}^{\text{ds}}(\mathcal{E}; \text{ChA}) = \max_{\mathcal{A}, M_0, M_1} 2 \Pr[\mathcal{A}(M_0, M_1, \text{ChA}(\mathcal{E}(M_b))) = b] - 1$$

where challenge bit b is uniformly distributed over $\{0, 1\}$ and the maximum is over all m -bit messages M_0, M_1 and all adversaries \mathcal{A} . For any metric \mathbf{xs} , we say \mathcal{E} provides \mathbf{XS} -security over ChA if $\lim_{k \rightarrow \infty} \mathbf{Adv}^{\mathbf{xs}}(\mathcal{E}; \text{ChA}) = 0$.

RELATIONS. The mutual information between message and ciphertext, as measured by *mis*, is, as noted above, the change in the number of bits needed to encode the message created by seeing the ciphertext. It is not clear why this should

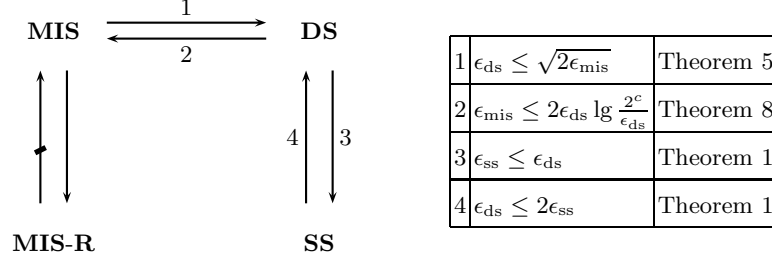


Fig. 2. Relations between notions. An arrow $\mathbf{A} \rightarrow \mathbf{B}$ is an implication, meaning every scheme that is \mathbf{A} -secure is also \mathbf{B} -secure, while a barred arrow $\mathbf{A} \not\rightarrow \mathbf{B}$ is a separation, meaning that there is a \mathbf{A} -secure scheme that is not \mathbf{B} -secure. If $\mathcal{E}: \{0, 1\}^m \rightarrow \{0, 1\}^c$ is the encryption algorithm and ChA the adversary channel, we let $\epsilon_{xs} = \text{Adv}^{xs}(\mathcal{E}; \text{ChA})$. The table then shows the quantitative bounds underlying the annotated implications.

measure privacy in the sense of semantic security. Yet we are able to show that mutual-information security and semantic security are equivalent, meaning an encryption scheme is MIS-secure if and only if it is SS-secure. Fig. 2 summarizes this and other relations we establish that between them settle all possible relations. The equivalence between SS and DS is the information-theoretic analogue of the corresponding well-known equivalence in the computational setting [19, 3]. As there, however, it brings the important benefit that we can now work with the technically simpler DS. We then show that MIS implies DS by reducing to Pinsker’s inequality. We show conversely that DS implies MIS via a general relation between mutual information and statistical distance. As Fig. 2 indicates, the asymptotic relations are all underlain by concrete quantitative and polynomial relations between the advantages. On the other hand, we show that in general MIS-R does not imply MIS, meaning the former is strictly weaker than the latter. We do this by exhibiting an encryption function \mathcal{E} and channel ChA such that \mathcal{E} is MIS-R-secure relative to ChA but MIS-insecure relative to ChA . Furthermore we do this for the case that ChA is a BSC.

OUR SCHEME. We provide the first explicit scheme with all the following characteristics over a large class of adversary and receiver channels including BSCs: (1) It is DS (hence SS, MIS and MIS-R) secure (2) It is proven to satisfy the decoding condition (3) It has optimal rate (4) It is fully polynomial time, meaning both encryption and decryption algorithms run in polynomial time (5) the errors in the security and decoding conditions do not just vanish in the limit but at an exponential rate. Our scheme is based on three main ideas: (1) the use of *invertible* extractors (2) analysis via smooth min-entropy, and (3) a (surprising) result saying that for certain types of schemes, security on random messages implies security on all messages.

Recall that the secrecy capacity is the optimal rate for MIS-R schemes meeting the decoding condition and in the case of BSCs it equals $h_2(p_A) - h_2(p_R)$.

Since DS-security is stronger than MIS-R security, the optimal rate could in principle be smaller. Perhaps surprisingly, it isn't: for a broad class of channels including symmetric channels, the optimal rate is the same for DS and MIS-R security. This follows by applying our above-mentioned result showing that MIS-R implies MIS for certain types of schemes and channels, to known results on achieving the secrecy capacity for MIS-R. Thus, when we say, above, that our scheme achieves optimal rate, this rate is in fact the secrecy capacity.

A common misconception is to think that privacy and error-correction may be completely de-coupled, meaning one would first build a scheme that is secure when the receiver channel is noiseless and then add an ECC on top to meet the decoding condition with a noisy receiver channel. This does not work because the error-correction helps the adversary by reducing the noise over the adversary channel. The two requirements do need to be considered together. Nonetheless, our approach is modular, combining (invertible) extractors with existing ECCs in a blackbox way. As a consequence, any ECC of sufficient rate may be used. This is an attractive feature of our scheme from the practical perspective. In addition our scheme is simple and efficient.

Our claims (proven DS-security and decoding with optimal rate) hold not only for BSCs but for a wide range of receiver and adversary channels.

A CONCRETE INSTANTIATION. As a consequence of our general paradigm, we prove that the following scheme achieves secrecy capacity for the BSC setting with $p_R < p_A \leq 1/2$. For any ECC $E: \{0, 1\}^k \rightarrow \{0, 1\}^n$ such that $k \approx (1 - h_2(p_R)) \cdot n$ (such ECCs can be built e.g. from polar codes [2] or from concatenated codes [18]) and a parameter $t \geq 1$, our encryption function \mathcal{E} , on input an m -bit message M , where $m = b \cdot t$ and $b \approx (h_2(p_A) - h_2(p_R)) \cdot n$, first selects a random k -bit string $A \neq 0^k$ and t random $(k - b)$ -bit strings $R[1], \dots, R[t]$. It then splits M into t b -bit blocks $M[1], \dots, M[t]$, and outputs

$$\mathcal{E}(M) = E(A) \parallel E(A \odot (M[1] \parallel R[1])) \parallel \dots \parallel E(A \odot (M[t] \parallel R[t])),$$

where \odot is multiplication of k -bit strings interpreted as elements of the extension field $\text{GF}(2^k)$.

RELATED WORK. This paper was formed by merging [5, 4] which together function as the full version of this paper. We refer there for all proofs omitted from this paper and also for full and comprehensive surveys of the large body of work related to wiretap security, and more broadly, to information-theoretically secure communication in a noisy setup. Here we discuss the most related work.

Relations between entropy- and distance-based security metrics have been explored in settings other than the wiretap one, using techniques similar to ours [13, 7, 15], the last in the context of statistically-private commitment. Iwamoto and Ohta [25] relate different notions of indistinguishability for statistically secure symmetric encryption. In the context of key-agreement in the wiretap setting (a simpler problem than ours) Csiszár [13] relates MIS-R and RDS, the latter being DS for random messages.

Wyner's syndrome coding approach [41] and extensions by Cohen and Zémor [9, 10] only provide weak security. Hayashi and Matsumoto [21] replace the

matrix-multiplication in these schemes by a universal hash function and show MIS-R security of their scheme. Their result could be used to obtain an alternative to the proof of RDS security used in our scheme for the common case where the extractor is obtained from a universal hash function. However, the obtained security bound is not explicit, making their result unsuitable to applications. Moreover, our proof also yields as a special case a cleaner proof for the result of [21].

Syndrome coding could be viewed as a special case of coset coding which is based on an inner code and outer code. Instantiations of this approach have been considered in [40, 33, 38] using LDPC codes, but polynomial-time decoding is possible only if the adversary channel is a binary erasure channel or the receiver channel is noiseless.

Mahdavi and Vardy [29] and Hof and Shamai [23] (similar ideas also appeared in [26, 1]) use polar codes [2] to build encryption schemes for the wiretap setting with binary-input symmetric channels. However, these schemes only provide weak security. The full version [30] of [29] provides a variant of the scheme achieving MIS-R security. They use results of the present paper (namely our above-mentioned result that MIS-R implies MIS for certain schemes, whose proof they re-produce for completeness), to conclude that their scheme is also MIS-secure. However there is no proof that decryption (decoding) is possible in their scheme, even in principle let alone in polynomial time. Also efficient generation of polar codes is an open research direction with first results only now appearing [39], and hence relying on this specific code family may be somewhat problematic. Our solution, in contrast, works for arbitrary codes.

As explained in [5], fuzzy extractors [17] can be used to build a DS-secure scheme with polynomial-time encoding and decoding, but the rate of this scheme is far from optimal and the approach is inherently limited to low-rate schemes. We note that (seedless) invertible extractors were previously used in [8] within schemes for the “wiretap channel II” model [34], where the adversary (adaptively) erases ciphertext bits. In contrast to our work, only random-message security was considered in [8].

2 Preliminaries

BASIC NOTATION AND DEFINITIONS. “PT” stands for “polynomial-time.” If s is a binary string then $s[i]$ denotes its i -th bit and $|s|$ denotes its length. If S is a set then $|S|$ denotes its size. If x is a real number then $|x|$ denotes its absolute value. A function $f : \{0, 1\}^m \rightarrow \{0, 1\}^n$ is *linear* if $f(x \oplus y) = f(x) \oplus f(y)$ for all $x, y \in \{0, 1\}^m$. A probability distribution is a function P that associates to each x a probability $P(x) \in [0, 1]$. The support $\text{SUPP}(P)$ is the set of all x such that $P(x) > 0$. All probability distributions in this paper are discrete. Associate to random variable X and event E the probability distributions $P_X, P_{X|E}$ defined for all x by $P_X(x) = \Pr[X = x]$ and $P_{X|E}(x) = \Pr[X = x \mid E]$. We denote by $\lg(\cdot)$ the logarithm in base two, and by $\ln(\cdot)$ the natural logarithm. We adopt standard conventions such as $0 \lg 0 = 0 \lg \infty = 0$

and $\Pr[E_1|E_2] = 0$ when $\Pr[E_2] = 0$. The function $h: [0, 1] \rightarrow [0, 1]$ is defined by $h(x) = -x \lg x$. The (Shannon) entropy of probability distribution P is defined by $\mathbf{H}(P) = \sum_x h(P(x))$ and the statistical difference between probability distributions P, Q is defined by $\mathbf{SD}(P; Q) = 0.5 \cdot \sum_x |P(x) - Q(x)|$. If X, Y are random variables the (Shannon) entropy is defined by $\mathbf{H}(X) = \mathbf{H}(P_X) = \sum_x h(P_X(x))$. The conditional entropy is defined via $\mathbf{H}(X|Y = y) = \sum_x h(P_{X|Y=y}(x))$ and $\mathbf{H}(X|Y) = \sum_y P_Y(y) \cdot \mathbf{H}(X|Y = y)$. The mutual information is defined by $\mathbf{I}(X; Y) = \mathbf{H}(X) - \mathbf{H}(X|Y)$. The statistical or variational distance between random variables X_1, X_2 is $\mathbf{SD}(X_1; X_2) = \mathbf{SD}(P_{X_1}; P_{X_2}) = 0.5 \cdot \sum_x |\Pr[X_1 = x] - \Pr[X_2 = x]|$. The min-entropy of random variable X is $\mathbf{H}_\infty(X) = -\lg(\max_x \Pr[X = x])$ and if Z is also a random variable the conditional min-entropy is $\mathbf{H}_\infty(X|Z) = -\lg(\sum_z \Pr[Z = z] \max_x \Pr[X = x|Z = z])$.

TRANSFORMS, CHANNELS AND ALGORITHMS. We say that T is a transform with domain D and range R , written $T: D \rightarrow R$, if $T(x)$ is a random variable over R for every $x \in D$. Thus, T is fully specified by a sequence $P = \{P_x\}_{x \in D}$ of probability distributions over R , where $P_x(y) = \Pr[T(x) = y]$ for all $x \in D$ and $y \in R$. We call P the distribution associated to T . This distribution can be specified by a $|D|$ by $|R|$ transition probability matrix W defined by $W[x, y] = P_x(y)$. A channel is simply a transform. This is a very general notion of a channel but it does mean channels are memoryless in the sense that two successive transmissions over the same channel are independent random variables. The transition probability matrix representation is the most common one in this case. A (randomized) algorithm is also a transform. Finally, an adversary too is a transform, and so is a simulator.

If $T: \{0, 1\} \rightarrow R$ is a transform we may apply it to inputs of any length. The understanding is that T is applied independently to each bit of the input. For example \mathbf{BSC}_p , classically defined as a 1-bit channel, is here viewed as taking inputs of arbitrary length and flipping each bit independently with probability p . Similarly, we apply a transform $T: \{0, 1\}^l \rightarrow R$ to any input whose length is divisible by l . We say that a transform $T: D \rightarrow R$ with transition matrix W is *symmetric* if there exists a partition of the range as $R = R_1 \cup \dots \cup R_n$ such that for all i the sub-matrix $W_i = W[\cdot, R_i]$ induced by the columns in R_i is strongly symmetric, i.e., all rows of W_i are permutations of each other, and all columns of W_i are permutations of each other.

3 Security metrics and relations

ENCRYPTION FUNCTIONS AND SCHEMES. An *encryption function* is a transform $\mathcal{E}: \{0, 1\}^m \rightarrow \{0, 1\}^c$ where m is the message length and c is the sender ciphertext length. The *rate* of \mathcal{E} is $\mathbf{Rate}(\mathcal{E}) = m/c$. If $\mathbf{ChR}: \{0, 1\}^c \rightarrow \{0, 1\}^d$ is a receiver channel then a *decryption function* for \mathcal{E} over \mathbf{ChR} is a transform $\mathcal{D}: \{0, 1\}^d \rightarrow \{0, 1\}^m$ whose decryption error $\mathbf{DE}(\mathcal{E}; \mathcal{D}; \mathbf{ChR})$ is defined as the maximum, over all $M \in \{0, 1\}^m$, of $\Pr[\mathcal{D}(\mathbf{ChR}(\mathcal{E}(M))) \neq M]$.

An *encryption scheme* $\bar{\mathcal{E}} = \{\mathcal{E}_k\}_{k \in \mathbb{N}}$ is a family of encryption functions where $\mathcal{E}_k: \{0, 1\}^{m(k)} \rightarrow \{0, 1\}^{c(k)}$ for functions $m, c: \mathbb{N} \rightarrow \mathbb{N}$ called the mes-

sage length and sender ciphertext length of the scheme. The *rate* of $\bar{\mathcal{E}}$ is the function $\mathbf{Rate}_{\bar{\mathcal{E}}}: \mathbb{N} \rightarrow \mathbb{R}$ defined by $\mathbf{Rate}_{\bar{\mathcal{E}}}(k) = \mathbf{Rate}(\mathcal{E}_k)$ for all $k \in \mathbb{N}$. Suppose $\overline{\mathbf{ChR}} = \{\mathbf{ChR}_k\}_{k \in \mathbb{N}}$ is a family of receiver channels where $\mathbf{ChR}_k: \{0, 1\}^{c(k)} \rightarrow \{0, 1\}^{d(k)}$. Then a *decryption scheme* for $\bar{\mathcal{E}}$ over $\overline{\mathbf{ChR}}$ is a family $\overline{\mathcal{D}} = \{\mathcal{D}_k\}_{k \in \mathbb{N}}$ where $\mathcal{D}_k: \{0, 1\}^{d(k)} \rightarrow \{0, 1\}^{m(k)}$ is a decryption function for \mathcal{E}_k over \mathbf{ChR}_k . We say that $\overline{\mathcal{D}}$ is a correct decryption scheme for $\bar{\mathcal{E}}$ relative to $\overline{\mathbf{ChR}}$ if the limit as $k \rightarrow \infty$ of $\mathbf{DE}(\mathcal{E}_k; \mathcal{D}_k; \mathbf{ChR}_k)$ is 0. We say that encryption scheme $\bar{\mathcal{E}}$ is decryptable relative to $\overline{\mathbf{ChR}}$ if there exists a correct decryption scheme for $\bar{\mathcal{E}}$ relative to $\overline{\mathbf{ChR}}$. A family $\{\mathcal{S}_k\}_{k \in \mathbb{N}}$ (eg. an encryption or decryption scheme) is PT if there is a polynomial time computable function which on input 1^k (the unary representation of k) and x returns $\mathcal{S}_k(x)$. Our constructs will provide PT encryption and decryption schemes.

SECURITY METRICS. Let $\mathcal{E}: \{0, 1\}^m \rightarrow \{0, 1\}^c$ be an encryption function and let $\mathbf{ChA}: \{0, 1\}^c \rightarrow \{0, 1\}^d$ be an adversary channel. Security depends only on these, not on the receiver channel. We now define semantic security (ss), distinguishing security (ds), random mutual-information security (mis-r) and mutual information security (mis). For each type of security $xs \in \{\text{ss}, \text{ds}, \text{mis-r}, \text{mis}\}$, we associate to $\mathcal{E}; \mathbf{ChA}$ a real number $\mathbf{Adv}^{xs}(\mathcal{E}; \mathbf{ChA})$. The smaller this number, the more secure is $\mathcal{E}; \mathbf{ChA}$ according to the metric in question. The security of an encryption function is quantitative, as captured by the advantage. We might measure it in bits, saying that $\mathcal{E}; \mathbf{ChA}$ has s bits of xs -security if $\mathbf{Adv}^{xs}(\mathcal{E}; \mathbf{ChA}) \leq 2^{-s}$. A qualitative definition of “secure,” meaning one under which something is secure or not secure, may only be made asymptotically, meaning for schemes. We say encryption scheme $\bar{\mathcal{E}} = \{\mathcal{E}_k\}_{k \in \mathbb{N}}$ is XS-secure relative to $\overline{\mathbf{ChA}} = \{\mathbf{ChA}_k\}_{k \in \mathbb{N}}$ if $\lim_{k \rightarrow \infty} \mathbf{Adv}^{xs}(\mathcal{E}_k; \mathbf{ChA}_k) = 0$. This does not mandate any particular rate at which the advantage should vanish, but in our constructions this rate is exponentially vanishing with k . We define the ss advantage $\mathbf{Adv}^{\text{ss}}(\mathcal{E}; \mathbf{ChA})$ as

$$\max_{f, M} \left(\max_{\mathcal{A}} \Pr[\mathcal{A}(\mathbf{ChA}(\mathcal{E}(M))) = f(M)] - \max_S \Pr[\mathcal{S}(m) = f(M)] \right). \quad (1)$$

Here f is a transform with domain $\{0, 1\}^m$ that represents partial information about the message. Examples include $f(M) = M$ or $f(M) = M[1]$ or $f(M) = M[1] \oplus \dots \oplus M[m]$, where $M[i]$ is the i -th bit of M . But f could be a much more complex function, and could even be randomized. The adversary’s goal is to compute $f(M)$ given an adversary ciphertext $\mathbf{ChA}(\mathcal{E}(M))$ formed by encrypting message M . The probability that it does this is $\Pr[\mathcal{A}(\mathbf{ChA}(\mathcal{E}(M))) = f(M)]$, then maximized over all adversaries \mathcal{A} to achieve strategy independence. We then subtract the a priori probability of success, meaning the maximum possible probability of computing $f(M)$ if you are not given the adversary ciphertext. Finally, the outer max over all f, M ensures that the metric measures the extent to which *any* partial information leaks regardless of message distribution. We define the distinguishing advantage via

$$\mathbf{Adv}^{\text{ds}}(\mathcal{E}; \mathbf{ChA}) = \max_{\mathcal{A}, M_0, M_1} 2 \Pr[\mathcal{A}(M_0, M_1, \mathbf{ChA}(\mathcal{E}(M_b))) = b] - 1 \quad (2)$$

$$= \max_{M_0, M_1} \mathbf{SD}(\mathbf{ChA}(\mathcal{E}(M_0)); \mathbf{ChA}(\mathcal{E}(M_1))). \quad (3)$$

In Eq. (2), $\Pr[\mathcal{A}(M_0, M_1, \text{ChA}(\mathcal{E}(M_b))) = b]$ is the probability that adversary \mathcal{A} , given m -bit messages M_0, M_1 and an adversary ciphertext emanating from M_b , correctly identifies the random challenge bit b . The a priori success probability being $1/2$, the advantage is appropriately scaled. This advantage is equal to the statistical distance between the random variables $\text{ChA}(\mathcal{E}(M_0))$ and $\text{ChA}(\mathcal{E}(M_1))$. The mutual-information security advantages are defined via

$$\mathbf{Adv}^{\text{mir-r}}(\mathcal{E}; \text{ChA}) = \mathbf{I}(\mathbf{U}; \text{ChA}(\mathcal{E}(\mathbf{U}))) \quad (4)$$

$$\mathbf{Adv}^{\text{mis}}(\mathcal{E}; \text{ChA}) = \max_{\mathbf{M}} \mathbf{I}(\mathbf{M}; \text{ChA}(\mathcal{E}(\mathbf{M}))) \quad (5)$$

where the random variable \mathbf{U} is uniformly distributed over $\{0, 1\}^m$.

DS IS EQUIVALENT TO SS. Theorem 1 below says that SS and DS are equivalent up to a small constant factor in the advantage. This is helpful because DS is more analytically tractable than SS. The proof is an extension of the classical ones in computational cryptography and is given in [5].

Theorem 1. [DS \leftrightarrow SS] *Let $\mathcal{E}: \{0, 1\}^m \rightarrow \{0, 1\}^c$ be an encryption function and ChA an adversary channel. Then $\mathbf{Adv}^{\text{ss}}(\mathcal{E}; \text{ChA}) \leq \mathbf{Adv}^{\text{ds}}(\mathcal{E}; \text{ChA}) \leq 2 \cdot \mathbf{Adv}^{\text{ss}}(\mathcal{E}; \text{ChA})$. ■*

MIS IMPLIES DS. The KL divergence is a distance measure for probability distributions P, Q defined by $\mathbf{D}(P; Q) = \sum_x P(x) \lg P(x)/Q(x)$. Let \mathbf{M}, \mathbf{C} be random variables. Probability distributions $J_{\mathbf{M}, \mathbf{C}}, I_{\mathbf{M}, \mathbf{C}}$ are defined for all M, C by $J_{\mathbf{M}, \mathbf{C}}(M, C) = \Pr[\mathbf{M} = M, \mathbf{C} = C]$ and $I_{\mathbf{M}, \mathbf{C}}(M, C) = \Pr[\mathbf{M} = M] \cdot \Pr[\mathbf{C} = C]$. Thus $J_{\mathbf{M}, \mathbf{C}}$ is the joint distribution of \mathbf{M} and \mathbf{C} , while $I_{\mathbf{M}, \mathbf{C}}$ is the “independent” or product distribution. The following is standard:

Lemma 2. *Let \mathbf{M}, \mathbf{C} be random variables. Then $\mathbf{I}(\mathbf{M}; \mathbf{C}) = \mathbf{D}(J_{\mathbf{M}, \mathbf{C}}; I_{\mathbf{M}, \mathbf{C}})$. ■*

Pinsker’s inequality —from [35] with the tight constant from [12]— lower bounds the KL divergence between two distributions in terms of their statistical distance:

Lemma 3. *Let P, Q be probability distributions. Then $\mathbf{D}(P; Q) \geq 2 \cdot \mathbf{SD}(P; Q)^2$. ■*

To use the above we need the following, whose proof is in [5]:

Lemma 4. *Let \mathbf{M} be uniformly distributed over $\{M_0, M_1\} \subseteq \{0, 1\}^m$. Let $g: \{0, 1\}^m \rightarrow \{0, 1\}^c$ be a transform and let $\mathbf{C} = g(\mathbf{M})$. Then $\mathbf{SD}(J_{\mathbf{M}, \mathbf{C}}; I_{\mathbf{M}, \mathbf{C}})$ equals $\mathbf{SD}(g(M_0); g(M_1))/2$. ■*

Combining the lemmas, we show the following in [5]:

Theorem 5. [MIS \rightarrow DS] *Let $\mathcal{E}: \{0, 1\}^m \rightarrow \{0, 1\}^c$ be an encryption function and ChA an adversary channel. Then $\mathbf{Adv}^{\text{ds}}(\mathcal{E}; \text{ChA}) \leq \sqrt{2 \cdot \mathbf{Adv}^{\text{mis}}(\mathcal{E}; \text{ChA})}$. ■*

DS IMPLIES MIS. The following general lemma from [5] bounds the difference in entropy between two distributions in terms of their statistical distance. It is a slight strengthening of [11, Theorem 16.3.2]. Similar bounds are provided in [22].

Lemma 6. *Let P, Q be probability distributions. Let $N = |\text{SUPP}(P) \cup \text{SUPP}(Q)|$ and $\epsilon = \mathbf{SD}(P; Q)$. Then $\mathbf{H}(P) - \mathbf{H}(Q) \leq 2\epsilon \cdot \lg(N/\epsilon)$. ■*

To exploit this, we define the *pairwise statistical distance* $\mathbf{PSD}(\mathbf{M}; \mathbf{C})$ between random variables \mathbf{M}, \mathbf{C} as the maximum, over all messages $M_0, M_1 \in \text{SUPP}(P_{\mathbf{M}})$, of $\mathbf{SD}(P_{\mathbf{C}|\mathbf{M}=M_0}; P_{\mathbf{C}|\mathbf{M}=M_1})$. The proof of the following is in [5].

Lemma 7. *Let \mathbf{M}, \mathbf{C} be random variables. Then $\mathbf{SD}(P_{\mathbf{C}}; P_{\mathbf{C}|\mathbf{M}=M}) \leq \mathbf{PSD}(\mathbf{M}; \mathbf{C})$ for any M . ■*

From this we conclude in [5] that DS implies MIS:

Theorem 8. [DS \rightarrow MIS] *Let $\mathcal{E}: \{0, 1\}^m \rightarrow \{0, 1\}^c$ be an encryption function and ChA an adversary channel. Let $\epsilon = \mathbf{Adv}^{\text{ds}}(\mathcal{E}; \text{ChA})$. Then $\mathbf{Adv}^{\text{mis}}(\mathcal{E}; \text{ChA}) \leq 2\epsilon \cdot \lg(2^c/\epsilon)$. ■*

The somewhat strange-looking form of the bound of Theorem 8 naturally raises the question of whether Lemma 6 is tight. The following says that it is up to a constant factor of 4. The proof is in [5].

Proposition 9. *Let $n > k \geq 1$ be integers. Let $\epsilon = 2^{-k}$ and $N = 1 + \epsilon 2^n$. Then there are distributions P, Q with $|\text{SUPP}(P) \cup \text{SUPP}(Q)| = N$ and $\mathbf{SD}(P; Q) = \epsilon$ and $\mathbf{H}(P) - \mathbf{H}(Q) \geq 0.5 \cdot \epsilon \cdot \lg(N/\epsilon)$. ■*

OTHER RELATIONS. We have now justified all the numbered implication arrows in Fig. 2. The un-numbered implication $\mathbf{MIS} \rightarrow \mathbf{MIS-R}$ is trivial. The intuition for the separation $\mathbf{MIS-R} \not\rightarrow \mathbf{MIS}$ is simple. Let \mathcal{E} be the identity function. Let ChA faithfully transmit inputs 0^m and 1^m and be very noisy on other inputs. Then \mathbf{MIS} fails because the adversary has high advantage when the message takes on only values $0^m, 1^m$ but $\mathbf{MIS-R}$ -security holds since these messages are unlikely. This example may seem artificial. In [5] we give a more complex example where ChA is a BSC and the encryption function is no longer trivial.

4 DS-Secure Encryption Achieving Secrecy Capacity

This section presents our main technical result, an encryption scheme achieving DS-security. Its rate, for a large set of adversary channels, is optimal.

HIGH-LEVEL APPROACH. We start by considering an extension of the usual setting where sender and receiver share a *public* random value S , i.e., known to the adversary, and which we call the *seed*. We will call an encryption function in this setting a *seeded encryption function*. For simplicity, this discussion will focus on the case where ChR and ChA are BSCs with respective crossover probabilities $p_R < p_A \leq 1/2$, and we assume that sender and receiver only want to agree on a joint secret key. If we let S be the seed of an extractor $\text{Ext}: \text{SDS} \times \{0, 1\}^k \rightarrow \{0, 1\}^m$ and given an error-correcting code $\mathbf{E}: \{0, 1\}^k \rightarrow \{0, 1\}^n$ for reliable communication over BSC_{p_R} , a natural approach consists of the sender sending $\mathbf{E}(R)$, for a random k -bit R , to the receiver, and both parties now derive the key as $K = \text{Ext}(S, R)$, since the receiver can recover R with very high probability.

The achievable key length is at most $\mathbf{H}_\infty(R|Z)$, where $Z = \text{BSC}_{p_A}(\mathbf{E}(R))$. Yet, it is not hard to see that the most likely outcome, when $Z = z$, is that R equals the unique r such that $\mathbf{E}(r) = z$, and that hence $\mathbf{H}_\infty(R|Z) = n \cdot \lg(1/(1 - p_A))$, falling short of achieving capacity $h(p_A) - h(p_R)$. To overcome this, we will observe the following: We can think of BSC_{p_A} as adding an n -bit vector E to its input $\mathbf{E}(R)$, where each bit $E[i]$ of the noise vector takes value one with probability p_A . With overwhelming probability, E is (roughly) uniformly distributed on the set of n -bit vectors with hamming weight (approximately) $p_A \cdot n$ and there are (approximately) $2^{n \cdot h_2(p_A)}$ such vectors. Therefore, choosing the noise uniformly from such vectors does not change the experiment much, and moreover, in this new experiment, one can show that roughly $\mathbf{H}_\infty(R|Z) \geq k - n \cdot (1 - h_2(p_A))$, which yields optimal rate using an optimal code with $k \approx (1 - h(p_R)) \cdot n$. We will make this precise for a general class of symmetric channels via the notion of *smooth min-entropy* [36].

But recall that our goal is way more ambitious: Alice wants to send an *arbitrary message of her choice*. The obvious approach is obtain a key K as above and then send an error-corrected version of $K \oplus M$. But this at least halves the rate, which becomes far from optimal. Our approach instead is to use an extractor Ext that is *invertible*, in the sense that given M and S , we can sample a random R such that $\text{Ext}(S, R) = M$. We then encrypt a message M as $\mathbf{E}(R)$, where R is a random preimage of M under $\text{Ext}(S, \cdot)$. However, the above argument only yields, at best, security for randomly chosen messages. In contrast, showing DS-security accounts to proving, for any two messages M_0 and M_1 , that $\text{BSC}_{p_A}(\mathbf{E}(R_0))$ and $\text{BSC}_{p_A}(\mathbf{E}(R_1))$ are statistically close, where R_i is uniform such that $\text{Ext}(S, R_i) = M_i$. To make things even worse, we allow the messages M_0 and M_1 are allowed to depend on the seed. The main challenge is that such proof appears to require detailed knowledge of the combinatorial structure of \mathbf{E} and Ext , as the actual ciphertext distribution depends on them.

Instead, we will take a completely different approach: We show that any seeded encryption function with appropriate linearity properties is DS-secure whenever it is secure for random messages. This result is surprising, as random-message security does *not*, in general, imply chosen-message security. A careful choice of the extractor to satisfy these requirements, combined with the above idea, yields a DS-secure seeded encryption function. The final step is to remove the seed, which is done by transmitting it (error-corrected) and amortizing out its impact on the rate to essentially zero by re-using it with the above seeded encryption function across blocks of the message. A hybrid argument is used to bound the decoding error and loss in security.

SEEDED ENCRYPTION. A *seeded encryption function* $\mathcal{SE}: \text{SDS} \times \{0, 1\}^b \rightarrow \{0, 1\}^n$ takes a seed $S \in \text{SDS}$ and message $M \in \{0, 1\}^b$ to return a sender ciphertext denoted $\mathcal{SE}(S, M)$ or $\mathcal{SE}_S(M)$; each seed S defines an encryption function $\mathcal{SE}_S: \{0, 1\}^b \rightarrow \{0, 1\}^n$. There must be a corresponding seeded decryption function $\mathcal{SD}: \text{SDS} \times \{0, 1\}^n \rightarrow \{0, 1\}^b$ such that $\mathcal{SD}(S, \mathcal{SE}(S, M)) = M$ for all S, M . We consider an extension of the standard wiretap setting where a seed $S \leftarrow_s \text{SDS}$ is a public parameter, available to sender, receiver and adversary. We extend DS-

transform $\mathcal{SE}(S, M)$: $// S \in \text{SDS}, M \in \{0, 1\}^b$ $R \leftarrow^* \{0, 1\}^r$; Ret $\mathbf{E}(\text{Inv}(S, R, M))$. transform $\mathcal{E}(M)$: $// M \in \{0, 1\}^m$ $S \leftarrow^* \text{SDS}$; $S[1], \dots, S[c] \xleftarrow{k} S$ $M[1], \dots, M[t] \xleftarrow{b} M$ For $i = 1$ to t do $C[i] \leftarrow^* \mathcal{SE}(S, M[i])$ Ret $\mathbf{E}(S[1]) \parallel \dots \parallel \mathbf{E}(S[c]) \parallel C[1] \parallel \dots \parallel C[t]$.	transform $\mathcal{D}(C)$: $// C \in \text{OUTR}^{(c+t)n}$ $C[1], \dots, C[c+t] \xleftarrow{n} C$ $S \leftarrow \mathbf{D}(C[1]) \parallel \dots \parallel \mathbf{D}(C[c])$ For $i = 1$ to t do $X[i] \leftarrow \mathbf{D}(C[c+i])$ $M[i] \leftarrow \text{Ext}(S, X[i])$ Ret $M[1] \parallel \dots \parallel M[t]$.
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Fig. 3. Encryption function $\mathcal{E} = \mathbf{RItE}_t[\text{Inv}, \mathbf{E}]$ using $\mathcal{SE} = \mathbf{ItE}[\text{Inv}, \mathbf{E}]$ and decryption function \mathcal{D} . By $X[1], \dots, X[c] \xleftarrow{b} X$ we mean that bc -bit string X is split into b -bit blocks.

security to this setting by letting $\mathbf{Adv}^{\text{ds}}(\mathcal{SE}; \text{ChA})$ be the expectation, over S drawn at random from SDS, of $\mathbf{Adv}^{\text{ds}}(\mathcal{SE}_S; \text{ChA})$. The rate of \mathcal{SE} is defined as b/n , meaning the seed is ignored.

EXTRACTORS. A function $\text{Ext}: \text{SDS} \times \{0, 1\}^k \rightarrow \{0, 1\}^b$ is an (h, α) -*extractor* if $\mathbf{SD}((\text{Ext}(S, X), Z, S); (U, Z, S)) \leq \alpha$ for all pairs of (correlated) random variables (X, Z) over $\{0, 1\}^k \times \{0, 1\}^*$ with $\mathbf{H}_\infty(X|Z) \geq h$, where additionally S and U are uniform on SDS and $\{0, 1\}^b$, respectively. (This is a strong, average case extractor in the terminology of [16].) We will say that Ext is *regular* if for all $S \in \text{SDS}$, the function $\text{Ext}(S, \cdot)$ is regular, meaning every point in the range has the same number of preimages.

INVERTING EXTRACTORS. We say that a function $\text{Inv}: \text{SDS} \times \{0, 1\}^r \times \{0, 1\}^b \rightarrow \{0, 1\}^k$ is an *inverter* for an extractor $\text{Ext}: \text{SDS} \times \{0, 1\}^k \rightarrow \{0, 1\}^b$ if for all $S \in \text{SDS}$ and $Y \in \{0, 1\}^b$, and for R uniform over $\{0, 1\}^k$, the random variable $\text{Inv}(S, R, Y)$ is uniformly distributed on $\{X \in \{0, 1\}^k : \text{Ext}(S, X) = Y\}$, the set of preimages of Y under $\text{Ext}(S, \cdot)$. To make this concrete we give an example of an extractor with an efficiently computable inverter. Recall that k -bit strings can be interpreted as elements of the finite field $\text{GF}(2^k)$, allowing us to define a multiplication operator \odot on k -bit strings. Then, for $\text{SDS} = \{0, 1\}^k \setminus 0^k$, we consider the function $\text{Ext}: \text{SDS} \times \{0, 1\}^k \rightarrow \{0, 1\}^b$ which, on inputs $S \in \text{SDS}$ and $X \in \{0, 1\}^k$, outputs the first b bits of $X \odot S$. It is easy to see that Ext is regular, as 0^k is not in the set of seeds. In [4] we prove the following using the average-case version of the Leftover Hash Lemma of [20], due to [16].

Lemma 10. *For all $\alpha \in (0, 1]$ and all $b \leq k - 2\lg(1/\alpha) + 2$ the function Ext is a $(b + 2\lg(1/\alpha) - 2, \alpha)$ -extractor.*

An efficient inverter $\text{Inv}: \text{SDS} \times \{0, 1\}^{k-b} \times \{0, 1\}^b \rightarrow \{0, 1\}^k$ is obtained via $\text{Inv}(S, R, M) = S^{-1} \odot (M \parallel R)$ where S^{-1} is the inverse of S with respect to multiplication in $\text{GF}(2^k)$.

THE RITE CONSTRUCTION. Let $\text{Ext} : \text{SDS} \times \{0,1\}^k \rightarrow \{0,1\}^b$ be a regular extractor with inverter $\text{Inv} : \text{SDS} \times \{0,1\}^r \times \{0,1\}^b \rightarrow \{0,1\}^k$. Also let $\text{E} : \{0,1\}^k \rightarrow \{0,1\}^n$ be an injective function with $k \leq n$, later to be instantiated via an ECC. Assume without loss of generality that for some $c \geq 1$, we have $|S| = c \cdot k$ for all $S \in \text{SDS}$. The encryption function \mathcal{E} is described in Fig. 3 and is obtained via the construction \mathbf{RItE}_t (Repeat Invert-then-Encode), where $t \geq 1$ is a parameter: As its main component, it relies on the construction \mathbf{ItE} (Invert-then-Encode) of a seeded encryption function $\mathbf{ItE}[\text{Inv}, \text{E}] : \text{SDS} \times \{0,1\}^b \rightarrow \{0,1\}^n$ which applies the inverter Inv to the message, and then applies E to the result. The final, seed-free, encryption function $\mathbf{RItE}_t[\text{Inv}, \text{E}]$ then takes an input $M \in \{0,1\}^m$, where $m = t \cdot b$, splits it into t b -bit blocks $M[1], \dots, M[t]$, chooses a random seed S , and combines an encoding of S with the encryptions of the blocks using \mathcal{SE}_S for $\mathcal{SE} = \mathbf{ItE}[\text{Inv}, \text{E}]$.

DECRYPTABILITY. Given a channel $\text{ChR} : \{0,1\} \rightarrow \text{OUTR}$, a decoder for E over ChR is a function $\text{D} : \text{OUTR}^n \rightarrow \{0,1\}^k$. Its decoding error is defined as $\mathbf{DE}(\text{E}; \text{D}; \text{ChR}) = \max_{M \in \{0,1\}^k} \Pr[\text{D}(\text{ChR}(\text{E}(M))) \neq M]$. Therefore, for any output alphabet OUTR and function $\text{D} : \text{OUTR}^n \rightarrow \{0,1\}^b$, we define the corresponding decryption function for \mathcal{E} over ChR as in Fig. 3. The following lemma summarizes the relation between its decryption error and the one of D .

Lemma 11. [Correct decryption] *Let $\text{ChR} : \{0,1\} \rightarrow \text{OUTR}$ be a channel, and let \mathcal{E} , D , E , and D be as above. Then, $\mathbf{DE}(\mathcal{E}; \text{D}; \text{ChR}) \leq (c + t) \cdot \mathbf{DE}(\text{E}; \text{D}; \text{ChR})$. ■*

STEP I: FROM RITE TO ITE. We reduce security of \mathbf{RItE} to that of \mathbf{ItE} . The proof of the following [4] uses a hybrid argument.

Lemma 12. *Let $t \geq 1$, $\mathcal{E} = \mathbf{RItE}_t[\text{Inv}, \text{E}]$ and $\mathcal{SE} = \mathbf{ItE}[\text{Inv}, \text{E}]$. For all $\text{ChA} : \{0,1\}^n \rightarrow \text{OUTA}$ we have $\mathbf{Adv}^{\text{ds}}(\mathcal{E}; \text{ChA}) \leq t \cdot \mathbf{Adv}^{\text{ds}}(\mathcal{SE}; \text{ChA})$. ■*

STEP II: RDS-SECURITY OF ITE. Towards determining the DS-security of \mathbf{ItE} we first address the seemingly simpler question of proving security under *random* messages. Specifically, for a seeded encryption function $\mathcal{SE} : \text{SDS} \times \{0,1\}^b \rightarrow \{0,1\}^n$, we define the rds advantage $\mathbf{Adv}^{\text{rds}}(\mathcal{SE}; \text{ChA})$ as the expectation of $\mathbf{SD}((\text{ChA}(\mathcal{SE}(S, \text{U})), \text{U}); (\text{ChA}(\mathcal{SE}(S, \text{U}')), \text{U}))$ where U and U' are uniformly chosen and independent b -bit messages, and the expectation is taken over the choice of the seed S . Exploiting the notion of ϵ -smooth min-entropy [36], the following, proven in [4], establishes RDS-security of \mathbf{ItE} :

Lemma 13. [RDS-security of ItE] *Let $\delta > 0$, let $\text{ChA} : \{0,1\} \rightarrow \text{OUTA}$ be a symmetric channel, let $\text{Inv} : \text{SDS} \times \{0,1\}^r \times \{0,1\}^b \rightarrow \{0,1\}^k$ be the inverter of a regular $(k - n \cdot (\lg(|\text{OUTA}|) - \mathbf{H}(\text{ChA}) + \delta), \alpha)$ -extractor, and let $\text{E} : \{0,1\}^k \rightarrow \{0,1\}^n$ be injective. Then, for $\mathcal{SE} = \mathbf{ItE}[\text{Inv}, \text{E}]$, we have*

$$\mathbf{Adv}^{\text{rds}}(\mathcal{SE}; \text{ChA}) \leq 2 \cdot 2^{-\frac{n\delta^2}{2 \lg^2(|\text{OUTA}|+3)}} + \alpha. \quad \blacksquare$$

STEP III: FROM RDS- TO DS-SECURITY. In contrast to RDS-security, proving DS-security of \mathbf{ItE} seems to require a better grasp of the combinatorial structure

of \mathbf{E} and \mathbf{Inv} . More concretely, think of any randomized (seeded) encryption function $\mathcal{SE} : \text{SDS} \times \{0, 1\}^b \rightarrow \{0, 1\}^n$ as a deterministic map $\mathcal{SE} : \text{SDS} \times \{0, 1\}^r \times \{0, 1\}^b \rightarrow \{0, 1\}^n$ (for some r), where the second argument takes the role of the random coins. We call \mathcal{SE} *separable* if $\mathcal{SE}(S, R, M) = \mathcal{SE}(S, R, 0^b) \oplus \mathcal{SE}(S, 0^r, M)$ for all $S \in \text{SDS}$, $R \in \{0, 1\}^r$, and $M \in \{0, 1\}^b$. Also, it is *message linear* if $\mathcal{SE}(S, 0^r, \cdot)$ is linear for all $S \in \text{SDS}$. The following is true for encryption functions with both these properties, and is proven in [4].

Lemma 14. [RDS \Rightarrow DS] *Let $\text{ChA} : \{0, 1\} \rightarrow \text{OUTA}$ be symmetric. If \mathcal{SE} is separable and message linear, then $\text{Adv}^{\text{ds}}(\mathcal{SE}; \text{ChA}) \leq 2 \cdot \text{Adv}^{\text{rds}}(\mathcal{SE}; \text{ChA})$. ■*

Coming back to \mathbf{ItE} , we say that $\mathbf{Inv} : \text{SDS} \times \{0, 1\}^r \times \{0, 1\}^b \rightarrow \{0, 1\}^k$ is *output linear* if $\mathbf{Inv}(S, 0^r, \cdot)$ is linear for all $S \in \text{SDS}$. Moreover, it is *separable* if $\mathbf{Inv}(S, R, M) = \mathbf{Inv}(S, R, 0^b) \oplus \mathbf{Inv}(S, 0^r, M)$ for all $S \in \text{SDS}$, $R \in \{0, 1\}^r$, and $M \in \{0, 1\}^b$. For example, the inverter for the above extractor based on finite-field multiplication is easily seen to be output linear and separable, by the linearity of the map $M \mapsto S^{-1} \odot M$.

SECURITY. If we instantiate $\mathbf{ItE}[\mathbf{Inv}, \mathbf{E}]$ so that \mathbf{Inv} is both output linear and separable, and we let \mathbf{E} be linear, the encryption function \mathcal{SE} is easily seen to be message linear and separable. The following theorem now follows immediately by combining Lemma 12, Lemma 14, and Lemma 13.

Theorem 15. [DS-security of \mathbf{RItE}] *Let $\delta > 0$ and $t \geq 1$. Also, let $\text{ChA} : \{0, 1\} \rightarrow \text{OUTA}$ be a symmetric channel, let $\mathbf{Inv} : \text{SDS} \times \{0, 1\}^r \times \{0, 1\}^b \rightarrow \{0, 1\}^k$ be the output-linear and separable inverter of a regular $(k - n \cdot (\lg(|\text{OUTA}|) - \mathbf{H}(\text{ChA}) + \delta), \alpha)$ -extractor, and let $\mathbf{E} : \{0, 1\}^k \rightarrow \{0, 1\}^n$ be linear and injective. Then, for $\mathcal{E} = \mathbf{RItE}_t[\mathbf{Inv}, \mathbf{E}]$, we have*

$$\text{Adv}^{\text{ds}}(\mathcal{E}; \text{ChA}) \leq 2t \cdot \left(2 \cdot 2^{-\frac{n\delta^2}{2\lg^2(|\text{OUTA}|+3)}} + \alpha \right). \quad \blacksquare$$

INSTANTIATING THE SCHEME. Recall that if $\text{ChA} : \{0, 1\}^l \rightarrow \text{OUTA}$ and $\text{ChR} : \{0, 1\}^l \rightarrow \text{OUTR}$ are symmetric channels, their secrecy capacity equals [27] $(\mathbf{H}(\mathbf{U}|\text{ChA}(\mathbf{U})) - \mathbf{H}(\mathbf{U}|\text{ChR}(\mathbf{U}))) / l$, for a uniform l -bit \mathbf{U} . Also, for a channel ChR , we denote its (Shannon) capacity as $\mathbf{C}(\text{ChR}) = \max_{\mathbf{X}} \mathbf{I}(\mathbf{X}; \text{ChR}(\mathbf{X})) / l$. We will need the following result (cf. e.g. [18] for a proof).

Lemma 16. [18] *For any $l \in \mathbb{N}$ and any channel $\text{ChR} : \{0, 1\}^l \rightarrow \text{OUTR}$, there is a family $\mathbf{E} = \{\mathbf{E}_s\}_{s \in \mathbb{N}}$ of linear encoding functions $\mathbf{E}_s : \{0, 1\}^{k(s)} \rightarrow \{0, 1\}^{n(s)}$ (where $n(s)$ is a multiple of l), with corresponding decoding functions $\mathbf{D}_s : \text{OUTR}^{n(s)/l} \rightarrow \{0, 1\}^{k(s)}$, such that (i) $\mathbf{DE}(\mathbf{E}_s; \mathbf{D}_s; \text{ChR}) = 2^{-\Theta(k(s))}$, (ii) $\lim_{s \rightarrow \infty} k(s)/n(s) = \mathbf{C}(\text{ChR})$, and (iii) \mathbf{E} and \mathbf{D} are PT computable. ■*

We now derive a scheme $\overline{\mathcal{E}} = \{\mathcal{E}_s\}_{s \in \mathbb{N}}$ achieving secrecy capacity for the most common case $\text{ChR} = \text{BSC}_{p_R}$ and $\text{ChA} = \text{BSC}_{p_A}$, where $0 \leq p_R < p_A \leq \frac{1}{2}$. We start with a family of codes $\{\mathbf{E}_s\}_{s \in \mathbb{N}}$ for BSC_{p_R} guaranteed to exist by Lemma 16, where $\mathbf{E}_s : \{0, 1\}^{k(s)} \rightarrow \{0, 1\}^{n(s)}$ and $\lim_{s \rightarrow \infty} k(s)/n(s) = 1 - h_2(p_R)$, or,

equivalently, there exists ν such that $\nu(s) = o(1)$ and $k(s) = (1 - h_2(p_R) - \nu(s)) \cdot n(s)$. Then, we let $\delta(s) = (2 \lg^2(5))^{1/2} \cdot n(s)^{-1/4}$ and $\alpha(s) = 2^{-n(s)^{1/2}}$, and use the finite-field based extractor $\text{Ext}_s : \{0, 1\}^{k(s)} \times \{0, 1\}^{k(s)} \rightarrow \{0, 1\}^{b(s)}$, where $b(s) = k(s) - n(s) \cdot (1 - h_2(p_A) + \delta(s)) + 2 \lg(\alpha) = (h_2(p_A) - h_2(p_R) - \nu(s) - \delta(s) - 2 \cdot n(s)^{-1/2}) \cdot n(s)$. We note that the resulting scheme is equivalent to the one described in the introduction (with $A = S^{-1}$). With these parameters,

$$\mathbf{Adv}^{\text{ds}}(\mathcal{E}_s; \text{BSC}_{p_A}) \leq 6 \cdot t(s) \cdot 2^{-\sqrt{n}}, \quad \mathbf{DE}(\mathcal{E}_s; \mathcal{D}_s; \text{BSC}_{p_R}) \leq (t(s) + 1) \cdot 2^{-\Theta(k(s))}$$

by Theorem 15 and Lemma 11, respectively. The rate of \mathcal{E}_s is

$$\mathbf{Rate}(\mathcal{E}_s) = \frac{t(s)}{t(s) + 1} \cdot \left(h_2(p_A) - h_2(p_R) - \nu(s) - \delta(s) - \frac{2}{\sqrt{n(s)}} \right).$$

Setting $t(s) = \lg(k(s))$ yields $\lim_{s \rightarrow \infty} \mathbf{Rate}(\mathcal{E}_s) = h_2(p_A) - h_2(p_R)$.

EXTENSIONS. The proof applies also for any pair of symmetric channels **ChR** and **ChA**, and the resulting rate is the secrecy capacity if the capacity of **ChA** : $\{0, 1\} \rightarrow \text{OUTA}$ is $\lg(|\text{OUTA}|) - \mathbf{H}(\text{ChA})$, which is the case if and only if a uniform input to **ChA** produces a uniform output. For other channels, such as *erasure channels* (where each bit is left unchanged with probability δ and mapped to an erasure symbol with probability $1 - \delta$) our technique still yields good schemes which, however, may fall short of achieving capacity. We also remark that the above presentation is constrained to single input-bit base channels for simplicity only. Our results can be extended to discrete memoryless channels with l -bit inputs for $l > 1$. For example, Lemma 13 extends to arbitrary symmetric channels **ChA** : $\{0, 1\}^l \rightarrow \text{OUTA}$, at the price of replacing n by n/l in the security bound and in the extractor's entropy requirement. In contrast, we do not know whether Lemma 14 applies to arbitrary symmetric channels with l -bit inputs, but it does, for instance, extend to any channel of the form $\text{ChA}(X) = X \oplus \mathbf{E}$, where \mathbf{E} is an l -bit string sampled according to an input-independent noise distribution.

Acknowledgments

Bellare was supported in part by NSF grants CCF-0915675, CNS-0904380 and CNS-1116800. Tessaro was supported in part by NSF grants CCF-0915675, CCF-1018064.

This material is based on research sponsored by DARPA under agreement numbers FA8750-11-C-0096 and FA8750-11-2-0225. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the U.S. Government.

References

1. M. Andersson, V. Rathi, R. Thobaben, J. Kliewer, and M. Skoglund. Nested polar codes for wiretap and relay channels. Available at arxiv.org/abs/1006.3573, 2010.
2. E. Arkan. Channel polarization: A method for constructing capacity achieving codes for symmetric binary-input memoryless channels. *IEEE Transactions on Information Theory*, 55(7):3051–3073, 2009.
3. M. Bellare, A. Desai, E. Jorjipii, and P. Rogaway. A concrete security treatment of symmetric encryption. In *38th FOCS*, pages 394–403. IEEE Computer Society Press, Oct. 1997.
4. M. Bellare and S. Tessaro. Polynomial-time, semantically-secure encryption achieving the secrecy capacity, Jan. 2012. Available as arxiv.org/abs/1201.3160 and Cryptology Eprint Archive Report 2012/022.
5. M. Bellare, S. Tessaro, and A. Vardy. A cryptographic treatment of the wiretap channel, Jan. 2012. Available as arxiv.org/abs/1201.2205 and Cryptology Eprint Archive Report 2012/15.
6. M. Bloch and J. Barros. *Physical-Layer Security: From Information Theory to Security Engineering*. Cambridge Academic Press, 2011.
7. M. Bloch and J. N. Laneman. On the secrecy capacity of arbitrary wiretap channels. In *Proceedings of the 46th Allerton Conference on Communications, Control, and Computing*, pages 818–825, Sep 2008.
8. M. Cheraghchi, F. Didier, and A. Shokrollahi. Invertible extractors and wiretap protocols. *IEEE Transactions on Information Theory*, 58(2):1254–1274, 2012.
9. G. Cohen and G. Zémor. The wiretap channel applied to biometrics. In *Proc. of the International Symposium on Information Theory and Applications*, 2004.
10. G. Cohen and G. Zémor. Syndrome coding for the wire-tap channel revisited. In *Proc. of the IEEE Information Theory Workshop (ITW '06)*, pages 33–36. IEEE, 2006.
11. T. M. Cover and J. A. Thomas. *Elements of Information Theory*. John Wiley and Sons, 1991.
12. I. Csiszár. Information-type measures of difference of probability distributions and indirect observations. *Studia Scientiarum Mathematicarum Hungarica*, 2:299–318, 1967.
13. I. Csiszár. Almost independence and secrecy capacity. *Problems of Information Transmission*, 32(1):40–47, 1996.
14. I. Csiszár and J. Körner. Broadcast channels with confidential messages. *IEEE Transactions on Information Theory*, 24(3):339–348, 1978.
15. I. Damgård, T. Pedersen, and B. Pfitzmann. Statistical secrecy and multibit commitments. *IEEE Transactions on Information Theory*, 44(3):1143–1151, 1998.
16. Y. Dodis, R. Ostrovsky, L. Reyzin, and A. Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM Journal on Computing*, 38(1):97–139, 2008.
17. Y. Dodis, L. Reyzin, and A. Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In C. Cachin and J. Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 523–540. Springer, May 2004.
18. I. Dumer. Concatenated codes and their multilevel generalizations. In *The Handbook of Coding Theory*, pages 1191–1988. Elsevier, 1998.
19. S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

20. J. Håstad, R. Impagliazzo, L. A. Levin, and M. Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.
21. M. Hayashi and R. Matsumoto. Construction of wiretap codes from ordinary channel codes. In *Proceedings of the 2010 IEEE International Symposium on Information Theory (ISIT 2010)*, pages 2538–2542. IEEE, 2010.
22. S. Ho and R. Yeung. The interplay between entropy and variational distance. *IEEE Transactions on Information Theory*, 56(12):5906–5929, 2010.
23. E. Hof and S. Shamai. Secrecy-achieving polar-coding. In *Proceedings of the IEEE Information Theory Workshop (ITW 2010)*. IEEE, 2010.
24. ICC 2011 workshop on physical-layer security, June 2011. Kyoto, Japan.
25. M. Iwamoto and K. Ohta. Security notions for information theoretically secure encryptions. In *Proceedings of the 2011 IEEE International Symposium on Information Theory (ISIT 2011)*, pages 1777–1781. IEEE, 2011.
26. O. Koçluoglu and H. ElGamal. Polar coding for secure transmission. In *Proceedings of the IEEE International Symposium on Personal Indoor and Mobile Radio Communication*, pages 2698–2703, 2010.
27. S. Leung-Yan-Cheong. On a special class of wire-tap channels. *IEEE Transactions on Information Theory*, 23(5):625–627, 1977.
28. Y. Liang, H. Poor, and S. Shamai. Information theoretic security. *Foundations and Trends in Communications and Information Theory*, 5(4):355–580, 2008.
29. H. Mahdaviifar and A. Vardy. Achieving the secrecy capacity of wiretap channels using polar codes. In *Proceedings of the 2010 IEEE International Symposium on Information Theory (ISIT 2010)*, pages 913 – 917. IEEE, 2010.
30. H. Mahdaviifar and A. Vardy. Achieving the secrecy capacity of wiretap channels using polar codes. *IEEE Transactions on Information Theory*, 57(10):6428–6443, 2011.
31. U. Maurer. The strong secret key rate of discrete random triples. In R. E. Blahut, editor, *Communication and Cryptography – Two Sides of One Tapestry*, pages 271–285. Kluwer, 1994.
32. U. M. Maurer and S. Wolf. Information-theoretic key agreement: From weak to strong secrecy for free. In B. Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 351–368. Springer, May 2000.
33. J. Muramatsu and S. Miyake. Construction of wiretap channel codes by using sparse matrices. In *Proc. of the IEEE Information Theory Workshop (ITW 2009)*, pages 105–109. IEEE, 2009.
34. L. H. Ozarow and A. D. Wyner. Wire-tap channel II. In T. Beth, N. Cot, and I. Ingemarsson, editors, *EUROCRYPT’84*, volume 209 of *LNCS*, pages 33–50. Springer, Apr. 1985.
35. M. S. Pinsker. *Information and information stability of random variables and processes*. Holden Day, San Francisco, 1964.
36. R. Renner and S. Wolf. Simple and tight bounds for information reconciliation and privacy amplification. In B. K. Roy, editor, *ASIACRYPT 2005*, volume 3788 of *LNCS*, pages 199–216. Springer, Dec. 2005.
37. C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423, 623–656, July, October 1948.
38. A. Suresh, A. Subramanian, A. Thangaraj, M. Bloch, and S.W. McLaughlin. Strong secrecy for erasure wiretap channels. In *Proc. of the IEEE Information Theory Workshop (ITW 2010)*. IEEE, 2010.
39. I. Tal and A. Vardy. How to construct polar codes. In *Proc. of the IEEE Information Theory Workshop (ITW 2010)*. IEEE, 2010.

- 40. A. Thangaraj, S. Dihidar, A. Calderbank, S. McLaughlin, and J. Merolla. Applications of LDPC codes to the wiretap channel. *IEEE Transactions on Information Theory*, 53(8):2933–2945, 2007.
- 41. A. D. Wyner. The wire-tap channel. *Bell Systems Tech. Journal*, 54(8):1355–1387, 1975.
- 42. Ziva corporation. <http://www.ziva-corp.com/>.

Multi-Instance Security and its Application to Password-Based Cryptography

Mihir Bellare¹, Thomas Ristenpart², and Stefano Tessaro³

¹ Department of Computer Science & Engineering, University of California San Diego, cseweb.ucsd.edu/~mihir/

² Department of Computer Sciences, University of Wisconsin - Madison, pages.cs.wisc.edu/~rist/

³ CSAIL, Massachusetts Institute of Technology, people.csail.mit.edu/tessaro/

Abstract. This paper develops a theory of multi-instance (mi) security and applies it to provide the first proof-based support for the classical practice of salting in password-based cryptography. Mi-security comes into play in settings (like password-based cryptography) where it is computationally feasible to compromise a single instance, and provides a second line of defense, aiming to ensure (in the case of passwords, via salting) that the effort to compromise all of some large number m of instances grows linearly with m . The first challenge is definitions, where we suggest LORX-security as a good metric for mi security of encryption and support this claim by showing it implies other natural metrics, illustrating in the process that even lifting simple results from the si setting to the mi one calls for new techniques. Next we provide a composition-based framework to transfer standard single-instance (si) security to mi-security with the aid of a key-derivation function. Analyzing password-based KDFs from the PKCS#5 standard to show that they meet our indistinguishability-style mi-security definition for KDFs, we are able to conclude with the first proof that per password salts amplify mi-security as hoped in practice. We believe that mi-security is of interest in other domains and that this work provides the foundation for its further theoretical development and practical application.

1 Introduction

This paper develops a theory of *multi-instance security* and applies it to support practices in password-based cryptography.

BACKGROUND. Password-based encryption (PBE) in practice is based on the PKCS#5 (equivalently, RFC 2898) standard [32]. It encrypts a message M under a password pw by picking a random s -bit *salt* sa , deriving a key $L \leftarrow \text{KD}(pw\|sa)$ and returning $C' \leftarrow C\|sa$ where $C \leftarrow \mathcal{E}(L, M)$. Here \mathcal{E} is a symmetric encryption scheme, typically an IND-CPA AES mode of operation, and key-derivation function (KDF) $\text{KD}: \{0,1\}^* \rightarrow \{0,1\}^n$ is the c -fold iteration $\text{KD} = H^c$ of a cryptographic hash function $H: \{0,1\}^* \rightarrow \{0,1\}^n$. However, passwords are often poorly chosen [29], falling within a set D called a “dictionary” that is small

enough to exhaust. A brute-force attack now recovers the target password pw (thereby breaking the ind-cpa security of the encryption) using cN hashes where $N = |D|$ is the size of the dictionary. Increasing c increases this effort, explaining the role of this iteration count, but c cannot be made too large without adversely impacting the performance of PBE.

Consider now m users, the i -th with password pw_i . If the salt is absent ($s = 0$), the number of hashes for the brute force attack to recover all m passwords remains around cN , but if s is large enough that salts are usually distinct, it rises to mcN , becoming prohibitive for large m . Salting, thus, aims to make the effort to compromise m target passwords scale linearly in m . (It has no effect on the security of encryption under any one, particular target password.)

NEW DIRECTIONS. This practice, in our view, opens a new vista in theoretical cryptography, namely to look at the multi-instance (mi) security of a scheme. We would seek metrics of security under which an adversary wins when it breaks all of m instances *but not if it breaks fewer*. This means that the mi security could potentially be much higher than the traditional single-instance (si) security. We would have security amplification.

Why do this? As the above discussion of password-based cryptography shows, there are settings where the computational effort t needed to compromise a single instance is feasible. Rather than give up, we provide a second line of defense. We limit the *scale* of the damage, ensuring (in the case of passwords, via the mechanism of salting) that the computational effort to compromise all of m instances is (around) tm and thus prohibitive for large m . We can't prevent the occasional illness, but we can prevent an epidemic.

We initiate the study of multi-instance security with a foundational treatment in two parts. The first part is agnostic to whether the setting is password-based or not, providing definitions for different kinds of mi-security of encryption and establishing relations between them, concluding with the message that what we call LORX-security is a good choice. The second part of our treatment focuses on password-based cryptography, providing a modular framework that proves mi-security of password-based primitives by viewing them as obtained by the composition of a mi-secure KDF with a si-secure primitive, and yielding in particular the first proof that salting works as expected to increase multi-instance security under a strong and formal metric for the latter.

Multi-instance security turns out to be challenging both definitionally (providing metrics where the adversary wins on breaking all instances but not fewer) and technically (reductions need to preserve tiny advantages and standard hybrid arguments no longer work). It also connects in interesting ways to security amplification via direct products and xor lemmas, eg. [37, 16, 19, 30, 13, 27, 34, 28, 35]. (We import some of their methods and export some novel viewpoints.) We believe there are many fruitful directions for future work, both theoretical (pursuing the connection with security amplification) and applied (mi security could be valuable in public-key cryptography where steadily improving attacks are making current security parameters look uncomfortably close to the edge for single-instance security). Let us now look at all this in some more detail.

LORX. We consider a setting with m independent target keys K_1, \dots, K_m . (They may, but need not, be passwords.) In order to show that mi-security grows with m we want a metric (definition) where the adversary wins if it breaks all m instances of the encryption but does not win if it breaks strictly fewer. If “breaking” is interpreted as recovery of the key then such a metric is easily given: it is the probability that the adversary recovers all m target keys. We refer to this as the UKU (Universal Key Unrecoverability) metric. But we know very well that key-recovery is a weak metric of encryption security. We want instead a mi analog of ind-cpa. The first thing that might come to mind is multi-user security [3, 2]. But in the latter the adversary wins (gets an advantage of one) even if it breaks just one instance so the mu-advantage of an adversary can never be less than its si (ind-cpa) advantage. We, in contrast, cannot “give up” once a single instance is broken. Something radically different is needed.

Our answer is LORX (left-or-right xor indistinguishability). Our game picks m independent challenge bits b_1, \dots, b_m and gives the adversary an oracle $\mathbf{Enc}(\cdot, \cdot, \cdot)$ which given i, M_0, M_1 returns an encryption of M_{b_i} under K_i . The adversary outputs a bit b' and its advantage is $2 \Pr[b' = b_1 \oplus \dots \oplus b_m] - 1$.⁴ Why xor? Its well-known “sensitivity” means that even if the adversary figures out $m - 1$ of the challenge bits, it will have low advantage unless it also figures out the last. This intuitive and historical support is strengthened by the relations, discussed below, that show that LORX implies security under other natural metrics.

RELATIONS. The novelty of multi-instance security prompts us to step back and consider a broad choice of definitions. Besides UKU and LORX, we define RORX (real-or-random xor indistinguishability, a mi-adaptation of the si ROR notion of [4]) and a natural AND metric where the challenge bits b_1, \dots, b_m and oracle $\mathbf{Enc}(\cdot, \cdot, \cdot)$ are as in the LORX game but the adversary output is a vector (b'_1, \dots, b'_m) and its advantage is $\Pr[(b'_1, \dots, b'_m) = (b_1, \dots, b_m)] - 2^{-m}$. The relations we provide, summarized in Figure 1, show that LORX emerges as the best choice because it implies all the others with tight reductions. Beyond that, they illustrate that the mi terrain differs from the si one in perhaps surprising ways, both in terms of relations and the techniques needed to establish them.

Thus, in the si setting, LOR and ROR are easily shown equivalent up to a factor 2 in the advantages [4]. It continues to be true that LORX easily implies RORX but the hybrid argument used to prove that ROR implies LOR [4] does not easily extend to the mi setting and the proof that RORX implies LORX is not only more involved but incurs a factor 2^m loss.⁵ In the si setting, both

⁴ This is a simplification of our actual definition, which allows the adversary to adaptively corrupt instances to reveal the underlying keys and challenge bits. This capability means that LORX-security implies threshold security where the adversary wins if it predicts the xor of the challenge bits of some subset of the instances of its choice. See Section 2 for further justification for this feature of the model.

⁵ This (exponential) 2^m factor loss is a natural consequence of the factor of 2 loss in the si case, our bound is tight, and the loss in applications is usually small because advantages are already exponentially vanishing in m . Nonetheless it is not always negligible and makes LORX preferable to RORX.

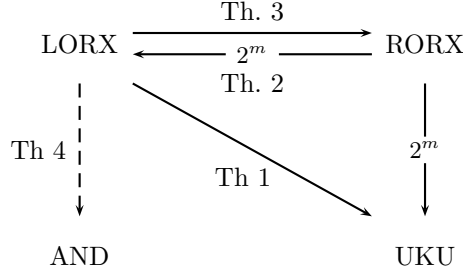


Fig. 1. Notions of multi-instance security for encryption and their relations. LORX (left-or-right xor indistinguishability) emerges as the strongest, tightly implying RORX (real-or-random xor indistinguishability) and UKU (universal key-unrecoverability). The dashed line indicates that under some (mild, usually met) conditions LORX also implies AND. RORX implies LORX and UKU but with a 2^m loss in advantage where m is the number of instances, making LORX a better choice.

LOR and ROR are easily shown to imply KU (key unrecoverability). Showing LORX implies UKU is more involved, needing a boosting argument to ensure preservation of exponentially-vanishing advantages. This reduction is tight but, interestingly, the reduction showing RORX implies UKU is not, incurring a 2^m -factor loss, again indicating that LORX is a better choice. We show that LORX usually implies AND by exploiting a direct product theorem by Unger [35], evidencing the connections with this area. Another natural metric of mi-security is a threshold one, but our incorporation of corruptions means that LORX implies security under this metric.

MI-SECURITY OF PBE. Under the LORX metric, we prove that the advantage ϵ' obtained by a time t adversary against m instances of the above PBE scheme \mathcal{E}' is at most $\epsilon + (q/mcN)^m$ (we are dropping negligible terms) where q is the number of adversary queries to RO H and ϵ is the advantage of a time t ind-cpa (si) adversary against \mathcal{E} . This is the desired result saying that salting works to provide a second line of defense under a strong mi security metric, amplifying security linearly in the number of instances.

FRAMEWORK. This result for PBE is established in a modular (rather than ad hoc) way, via a framework that yields corresponding results for any password-based primitive. This means not only ones like password-based message authentication (also covered in PKCS#5) or password-based authenticated encryption (WinZip) but public-key primitives like password-based digital signatures, where the signing key is derived from a password. We view a password-based scheme for a goal as derived by composing a key-derivation function (KDF) with a standard (si) scheme for the same goal. The framework then has the following components. (1) We provide a definition of mi-security for KDFs. (2) We provide composition theorems, showing that composing a mi-secure KDF with a si-secure scheme for a goal results in a mi-secure scheme for that goal. (We will illustrate this for

the case of encryption but similar results may be shown for other primitives.)
 (3) We analyze the iterated hash KDF of PKCS#5 and establish its mi security.

The statements above are qualitative. The quantitative security aspect is crucial. The definition of mi-security of KDFs must permit showing mi-security much higher than si-security. The reductions in the composition theorems must preserve exponentially vanishing mi-advantages. And the analysis of the PKCS#5 KDF must prove that the adversary advantage in q queries to the RO H grows as $(q/cmN)^m$, not merely q/cN . These quantitative constraints represent important technical challenges.

MI-SECURITY OF KDFS. We expand on item (1) above. The definition of mi-security we provide for KDFs is a simulation-based one inspired by the indistinguishability framework [26, 11]. The attacker must distinguish between the real world and an ideal counterpart. In both, target passwords pw_1, \dots, pw_m and salts sa_1, \dots, sa_m are randomly chosen. In the real world, the adversary gets input $(pw_1, sa_1, \text{KD}(pw_1 \| sa_1)), \dots, (pw_m, sa_m, \text{KD}(pw_m \| sa_1))$ and also gets an oracle for the RO hash function H used by KD. In the ideal world, the input is $(pw_1, sa_1, L_1), \dots, (pw_m, sa_m, L_m)$ where the keys L_1, \dots, L_m are randomly chosen, and the oracle is a simulator. The simulator itself has access to a **Test** oracle that will take a guess for a password and tell the simulator whether or not it matches one of the target passwords. Crucially, we require that when the number of queries made by the adversary to the simulator is q , the number of queries made by the simulator to its **Test** oracle is only q/c . This restriction is critical to our proof of security amplification and a source of challenges therein.

RELATED WORK. Previous work which aimed at providing proof-based assurances for password-based key-derivation has focused on the single-instance case and the role of iteration as represented by the iteration count c . Our work focuses on the multi-instance case and the roles of both salting and iteration.

The UNIX password hashing algorithm maps a password pw to $E_{pw}^c(0)$ where E is a blockcipher and 0 is a constant. Luby and Rackoff [24] show this is a one-way function when $c = 1$ and pw is a random blockcipher key. (So their result does not really cover passwords.) Wagner and Goldberg [36] treat the more general case of arbitrary c and keys that are passwords, but the goal continues to be to establish one-wayness and no security amplification (meaning increase in security with c) is shown. Boyen [8, 9] suggests various ways to enhance security, including letting users pick their own iteration counts.

Yao and Yin [38] give a natural pseudorandomness definition of a KDF in which the attacker gets (K, sa) where K is either $H^c(pw \| sa)$ or a random string of the same length and must determine which. Modeling H as a random oracle (RO) [7] to which the adversary makes q queries, they claim to prove that the adversary's advantage is at most q/cN plus a negligible term. This would establish single-instance security amplification by showing that iteration works as expected to increase attacker effort.⁶ However, even though salts are considered,

⁶ Unfortunately, we point in [6] to a bug in the proof of [38, Lemma 2.2] and explain why the bound claimed by [38, Theorem 1] is wrong. Beyond this, the proof makes

this does not consider multi-instance security let alone establish multi-instance security amplification, and their definition of KDF security does not adapt to allow this. (We use, as indicated above, an indistinguishability-style definition.) In fact the KDF definition of [38] is not even sufficient to establish si security of password-based encryption in the case the latter, as specified in PKCS#5, picks a fresh salt for each message encrypted. Kelsey, Schneier, Hall and Wagner [21] look into the time for password-recovery attacks for different choices of KDFs.

KDFs are for use in non-interactive settings like encryption with WinZip. The issues and questions we consider do not arise with password authenticated key exchange (PAKE) [5, 10, 14] where definitions already guarantee that the session key may be safely used for encryption. There are no salts and no amplification issues. Abadi and Warinschi [1] provide a si, key-recovery definition for PBE security and connect this with symbolic notions. They do not consider mi security. Dodis, Gennaro, Håstad, Krawczyk and Rabin [12] treat statistically-secure key derivation using hash functions and block ciphers. As discussed in-depth by Krawczyk [23], these results and techniques aren't useful for password-based KDFs because passwords aren't large enough, let alone have the sufficient amount of min-entropy. Krawczyk [23] also notes that his two-stage KDF approach could be used to build password-based KDFs by replacing the extraction stage with a key-stretching operation. Our general framework may be used to analyze the mi-security of this construction.

Work on direct product theorems and XOR lemmas (eg. [37, 15, 18, 13, 27]) has considered the problem of breaking multiple instances of a cryptographic primitive, in general as an intermediate step to amplifying security in the single-instance setting. Mi-Xor-security is used in this way in [13, 27].

2 The Multi-Instance Terrain

This section defines metrics of mi-secure encryption and explores the relations between them to establish the notions and results summarized in Figure 1. Our treatment intends to show that the mi terrain is different from the si one in fundamental ways, leading to new definitions, challenges and connections.

SYNTAX. Recall that a symmetric encryption scheme is a triple of algorithms $SE = (\mathcal{K}, \mathcal{E}, \mathcal{D})$. The key generation algorithm \mathcal{K} outputs a key. The encryption algorithm \mathcal{E} takes a key K and a message M and outputs a ciphertext $C \leftarrow_s \mathcal{E}(K, M)$. The deterministic decryption algorithm \mathcal{D} takes K and a ciphertext C to return either a string or \perp . Correctness requires that $\mathcal{D}(K, \mathcal{E}(K, M)) = M$ for all M with probability 1 over $K \leftarrow_s \mathcal{K}$ and the coins of \mathcal{E} .

To illustrate the issues and choices in defining mi security, we start with key unrecoverability which is simple because it is underlain by a computational game and its mi counterpart is easily and uncontentiously defined. When we move to

some rather large and not fully justified jumps. The special case $m = 1$ of our treatment will fill these gaps and recover the main claim of [38].

main $\text{UKU}_{\text{SE},m}^{\mathcal{A}}$ $\mathbf{K}[1], \dots, \mathbf{K}[m] \leftarrow \$ \mathcal{K}; \mathbf{K}' \leftarrow \$ \mathcal{A}^{\text{Enc}}$ Ret $\mathbf{K}' = \mathbf{K}$		proc. $\text{Enc}(i, M)$ Ret $\mathcal{E}(\mathbf{K}[i], M)$	proc. $\text{Cor}(i)$ Ret $\mathbf{K}[i]$
main $\text{LORX}_{\text{SE},m}^{\mathcal{A}}$ $\mathbf{K}[1], \dots, \mathbf{K}[m] \leftarrow \$ \mathcal{K}$ $\mathbf{b} \leftarrow \$ \{0, 1\}^m$ $b' \leftarrow \$ \mathcal{A}^{\text{Enc}}$ Ret $(b' = \oplus_i \mathbf{b}[i])$	main $\text{AND}_{\text{SE},m}^{\mathcal{A}}$ $\mathbf{K}[1], \dots, \mathbf{K}[m] \leftarrow \$ \mathcal{K}$ $\mathbf{b} \leftarrow \$ \{0, 1\}^m$ $\mathbf{b}' \leftarrow \$ \mathcal{A}^{\text{Enc}}$ Ret $(\mathbf{b}' = \mathbf{b})$	proc. $\text{Enc}(i, M_0, M_1)$ If $ M_0 \neq M_1 $ then Ret \perp $C \leftarrow \$ \mathcal{E}(\mathbf{K}[i], M_{\mathbf{b}[i]})$ Ret C	proc. $\text{Cor}(i)$ Ret $(\mathbf{K}[i], \mathbf{b}[i])$
main $\text{RORX}_{\text{SE},m}^{\mathcal{A}}$ $\mathbf{K}[1], \dots, \mathbf{K}[m] \leftarrow \$ (\{0, 1\}^k)^m$ $\mathbf{b} \leftarrow \$ \{0, 1\}^m; b' \leftarrow \$ \mathcal{A}^{\text{Enc}}$ Ret $(b' = \oplus_i \mathbf{b}[i])$	proc. $\text{Enc}(i, M)$ $C_1 \leftarrow \$ \mathcal{E}(\mathbf{K}[i], M)$ $M_0 \leftarrow \$ \{0, 1\}^{ M }; C_0 \leftarrow \$ \mathcal{E}(\mathbf{K}[i], M_0)$ Ret $C_{\mathbf{b}[i]}$	proc. $\text{Cor}(i)$ Ret $(\mathbf{K}[i], \mathbf{b}[i])$	

Fig. 2. Multi instance security notions for encryption.

stronger notions underlain by decisional games, definitions will get more difficult and more contentious as more choices will emerge.

UKU. Single-instance key unrecoverability is formalized via the game KU_{SE} where a target key $K \leftarrow \$ \mathcal{K}$ is initially sampled, and the adversary \mathcal{A} is given an oracle **Enc** which, on input M , returns $\mathcal{E}(K, M)$. Finally, the adversary is asked to output a guess K' for the key, and the game returns **true** if $K = K'$, and **false** otherwise. An mi version of the game, $\text{UKU}_{\text{SE},m}$, is depicted in Figure 2. It picks an m -vector \mathbf{K} of target keys and the oracle **Enc** now takes i, M to return $\mathcal{E}(\mathbf{K}[i], M)$. The **Cor** oracle gives the adversary the capability of corrupting a user to obtain its target key. The adversary's output guess is also a m -vector \mathbf{K}' and the game returns the boolean $(\mathbf{K} = \mathbf{K}')$, meaning the adversary wins only if it recovers *all* the target keys. (The “U” in “UKU” reflects this, standing for “Universal.”) The advantage of adversary \mathcal{A} is $\text{Adv}_{\text{SE},m}^{\text{uku}}(\mathcal{A}) = \Pr[\text{UKU}_{\text{SE},m}^{\mathcal{A}} \Rightarrow \text{true}]$. Naturally, this advantage depends on the adversary's resources. (It could be 1 if the adversary corrupts all instances.) We say that \mathcal{A} is a (t, \mathbf{q}, q_c) -adversary if it runs in time t and makes at most $\mathbf{q}[i]$ encryption queries of the form **Enc** (i, \cdot) and makes at most q_c corruption queries. Then we let $\text{Adv}_{\text{SE},m}^{\text{uku}}(t, \mathbf{q}, q_c) = \max_{\mathcal{A}} \text{Adv}_{\text{SE},m}^{\text{uku}}(\mathcal{A})$ where the maximum is over all (t, \mathbf{q}, q_c) -adversaries.

AND. Single-instance indistinguishability for symmetric encryption is usually formalized via left-or-right security [4]. A random bit b and key $K \leftarrow \$ \mathcal{K}$ are chosen, and an adversary \mathcal{A} is given access to an oracle **Enc** that given equal-length messages M_0, M_1 returns $\mathcal{E}(K, M_b)$. The adversary outputs a bit b' and its advantage is $2 \Pr[b = b'] - 1$. There are several ways one might consider creating an mi analog. Let us first consider a natural AND-based metric based on game $\text{AND}_{\text{SE},m}$ of Figure 2. It picks at random a vector $\mathbf{b} \leftarrow \$ \{0, 1\}^m$ of challenge bits as well as a vector $\mathbf{K}[1], \dots, \mathbf{K}[m]$ of keys, and the adversary is given access to

oracle **Enc** that on input i, M_0, M_1 , where $|M_0| = |M_1|$, returns $\mathcal{E}(\mathbf{K}[i], M_{\mathbf{b}[i]})$. Additionally, the corruption oracle **Cor** takes i and returns the pair $(\mathbf{K}[i], \mathbf{b}[i])$. The adversary finally outputs a bit vector \mathbf{b}' , and wins if and only if $\mathbf{b} = \mathbf{b}'$. (It is equivalent to test that $\mathbf{b}[i] = \mathbf{b}'[i]$ for all uncorrupted i .) The advantage of adversary \mathcal{A} is $\text{Adv}_{\text{SE},m}^{\text{and}}(\mathcal{A}) = \Pr[\text{AND}_{\text{SE},m}^{\mathcal{A}} \Rightarrow \text{true}] - 2^{-m}$. We say that \mathcal{A} is a (t, \mathbf{q}, q_c) -adversary if it runs in time t and makes at most $\mathbf{q}[i]$ encryption queries of the form **Enc** (i, \cdot, \cdot) and makes at most q_c corruption queries. Then we let $\text{Adv}_{\text{SE},m}^{\text{and}}(t, \mathbf{q}, q_c) = \max_{\mathcal{A}} \text{Adv}_{\text{SE},m}^{\text{and}}(\mathcal{A})$ where the maximum is over all (t, \mathbf{q}, q_c) -adversaries.

This metric has many points in its favor. By (later) showing that security under it is implied by security under our preferred LORX metric, we automatically garner whatever value it offers. But the AND metric also has weaknesses that in our view make it inadequate as the primary choice. Namely, it does not capture the hardness of breaking *all* the uncorrupted instances. For example, an adversary that corrupts instances $1, \dots, m-1$ to get $\mathbf{b}[1], \dots, \mathbf{b}[m-1]$, makes a random guess g for $\mathbf{b}[m]$ and returns $(\mathbf{b}[1], \dots, \mathbf{b}[m-1], g)$ has the high advantage $0.5 - 2^{-m}$ without breaking all instances. We prefer a metric where this adversary's advantage is close to 0.

LORX. To overcome the above issue with the AND advantage, we introduce the XOR advantage measure and use it to define LORX. Game $\text{LORX}_{\text{SE},m}$ of Figure 2 makes its initial choices the same way as game $\text{AND}_{\text{SE},m}$ and provides the adversary with the same oracles. However, rather than a vector, the adversary must output a bit b' , and wins if this equals $\mathbf{b}[1] \oplus \dots \oplus \mathbf{b}[m]$. (It is equivalent to test that $b' = \oplus_{i \in S} \mathbf{b}[i]$ where S is the uncorrupted set.) The advantage of adversary \mathcal{A} is $\text{Adv}_{\text{SE},m}^{\text{lorx}}(\mathcal{A}) = 2 \Pr[\text{LORX}_{\text{SE},m}^{\mathcal{A}} \Rightarrow \text{true}] - 1$. We say that \mathcal{A} is a (t, \mathbf{q}, q_c) -adversary if it runs in time t and makes at most $\mathbf{q}[i]$ encryption queries of the form **Enc** (i, \cdot, \cdot) and makes at most q_c corruption queries. Then we let $\text{Adv}_{\text{SE},m}^{\text{lorx}}(t, \mathbf{q}, q_c) = \max_{\mathcal{A}} \text{Adv}_{\text{SE},m}^{\text{lorx}}(\mathcal{A})$ where the maximum is over all (t, \mathbf{q}, q_c) -adversaries. In the example we gave for AND, if an adversary corrupts the first $m-1$ instances to get back $\mathbf{b}[1], \dots, \mathbf{b}[m-1]$, makes a random guess g for $\mathbf{b}[m]$ and outputs $b' = \mathbf{b}[1] \oplus \dots \oplus \mathbf{b}[m-1] \oplus g$, it will have advantage 0.

RORX. A variant of the si LOR notion, ROR, was given in [4]. Here the adversary must distinguish between an encryption of a message M it provides and the encryption of a random message of length $|M|$. This was shown equivalent to LOR up to a factor 2 in the advantages [4]. This leads us to define the mi analog RORX and ask how it relates to LORX. Game $\text{RORX}_{\text{SE},m}$ of Figure 2 makes its initial choices the same way as game $\text{LORX}_{\text{SE},m}$. The adversary is given access to oracle **Enc** that on input i, M , returns $\mathcal{E}(\mathbf{K}[i], M)$ if $\mathbf{b}[i] = 1$ and otherwise returns $\mathcal{E}(\mathbf{K}[i], M_1)$ where $M_1 \leftarrow_{\$} \{0, 1\}^{|M|}$. It also gets the usual **Cor** oracle. It outputs a bit b' and wins if this equals $\mathbf{b}[1] \oplus \dots \oplus \mathbf{b}[m]$. The advantage of adversary \mathcal{A} is $\text{Adv}_{\text{SE},m}^{\text{rorx}}(\mathcal{A}) = 2 \Pr[\text{RORX}_{\text{SE},m}^{\mathcal{A}} \Rightarrow \text{true}] - 1$. We say that \mathcal{A} is a (t, \mathbf{q}, q_c) -adversary if it runs in time t and makes at most $\mathbf{q}[i]$ encryption queries of the form **Enc** (i, \cdot) and makes at most q_c corruption queries. Then we let $\text{Adv}_{\text{SE},m}^{\text{rorx}}(t, \mathbf{q}, q_c) = \max_{\mathcal{A}} \text{Adv}_{\text{SE},m}^{\text{rorx}}(\mathcal{A})$ where the maximum is over all (t, \mathbf{q}, q_c) -adversaries.

DISCUSSION. The multi-user security goal from [3] gives rise to a version of the above games without corruptions and where all instances share the same challenge bit b , which the adversary tries to guess. But this does *not* measure mi security, since recovering a single key suffices to learn b .

The above approach extends naturally to providing a mi counterpart to any security definition based on a decisional game, where the adversary needs to guess a bit b . For example we may similarly create mi metrics of CCA security.

Why does the model include corruptions? The following example may help illustrate. Suppose SE is entirely insecure when the key has first bit 0 and highly secure otherwise. (From the si perspective, it is insecure.) In the LORX game, an adversary will be able to figure out around half the challenge bits. If we disallow corruptions, it would still have very low advantage. From the application point of view, this seems to send the wrong message. We want LORX-security to mean that the probability of “large scale” damage is low. But breaking half the instances is pretty large scale. Allowing corruptions removes this defect because the adversary could corrupt the instances it could not break and then, having corrupted only around half the instances, get a very high advantage, breaking LORX-security. In this way, we may conceptually keep the focus on an adversary goal of breaking *all* instances, yet cover the case of breaking some threshold number via the corruption capability.

An alternative way to address the above issue without corruptions is to define threshold metrics where the adversary wins by outputting a dynamically chosen set S and predicting the xor of the challenge bits for the indexes in S . This, again, has much going for it as a metric. But LORX with corruptions, as we define it, will imply security under this metric.

LORX IMPLIES UKU. In the si setting, it is easy to see that LOR security implies KU security. The LOR adversary simply runs the KU adversary. When the latter makes oracle query M , the LOR adversary queries its own oracle with M, M and returns the outcome to the KU adversary. When the latter returns a key K' , the LOR adversary submits a last oracle query consisting of a pair M_0, M_1 of random messages to get back a challenge ciphertext C , returning 1 if $\mathcal{D}(K', C) = M_1$ and 0 otherwise. A similar but slightly more involved proof shows that ROR implies KU.

It is important to establish analogs of these basic results in the mi setting, for they function as “tests” for the validity of our mi notions. The following shows that LORX security implies UKU. Interestingly, it is not as simple to establish in the mi case as in the si case. Also, as we will see later, the proof that RORX implies UKU is not only even more involved but incurs a factor 2^m loss, making LORX a better choice as the metric to target in designs.

Theorem 1. [LORX \Rightarrow UKU] *Let $\text{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme with message space \mathcal{M} , and let ℓ be such that $\{0, 1\}^\ell \subseteq \mathcal{M}$. Then, for all t , q_c , and \mathbf{q} , and for all $k \geq 1$,*

$$\text{Adv}_{\text{SE}, m}^{\text{uku}}(t, \mathbf{q}, q_c) \leq \text{Adv}_{\text{SE}, m}^{\text{lorx}}(t', \mathbf{q}', q_c) + m \cdot \left(\frac{1}{2^\ell - 1} \right)^k,$$

where $t' = t + O(m \cdot k)$, and $\mathbf{q}'[i] = \mathbf{q}[i] + k$ for all $i = 1, \dots, m$. ■

The proof is given in [6]. Here, let us stress Theorem 1 surfaces yet another subtlety of the mi setting. At first, it would seem that proving the case $k = 1$ of the theorem is sufficient (this is what usually done in the si case). However, it is crucial to remark that $\text{Adv}_{\text{SE},m}^{\text{lorx}}(t', \mathbf{q}', q_c)$ may be *very* small. For example, it is not unreasonable to expect $2^{-128 \cdot m}$ if SE is secure in the single-instance setting. Yet, assume that \mathcal{E} encrypts 128-bit messages, then we are only able to set $\ell = 128$, in turn making $m/(2^\ell - 1) \approx m \cdot 2^{-128}$ by far the leading term on the right-hand side. The parameter k hence opens the door to fine tuning of the additive extra term at the cost of an additive complexity loss in the reduction. Also note that the reduction in the proof of Theorem 1 is not immediate, as an adversary guessing all the keys in the UKU game with probability ϵ only yields an adversary recovering all the bits $\mathbf{b}[1], \dots, \mathbf{b}[m]$ in the LORX game with probability ϵ . Just outputting the xor of these bits is not sufficient, as we have to boost the success probability to $\frac{1+\epsilon}{2}$ in order to obtain the desired relation between the two advantage measures.

In analogy to the si setting, UKU does not imply LORX. Just take a scheme $\text{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ encrypting n -bit messages which is UKU-secure, and modify it into a scheme $\text{SE}' = (\mathcal{K}', \mathcal{E}', \mathcal{D}')$ where $\mathcal{K} = \mathcal{K}'$ and $\mathcal{E}'(K, M) = \mathcal{E}(K, M) \parallel M[0]$, with $M[0]$ being the first bit of M . Clearly, SE' is still UKU-secure but not LORX-secure.

As indicated above, a proof that RORX implies UKU is much more involved and incurs a factor 2^m loss. Roughly speaking, this is because in the si case, in the reduction needed to prove that ROR implies KU, the ROR adversary can only simulate the execution of the KU adversary correctly in the case where the bit is 1, i.e., the encryption oracle returns the actual encryption of a message. This results in a factor two loss in terms of advantage. Upon translating this technique to the mi case, the factor 2 becomes 2^m , as *all* bits need to be 1 for the UKU adversary to output the right keys with some guaranteed probability. However, we will not follow this route for the proof of this result. Instead, we can obtain the same result by combining Theorem 2 and Theorem 1.

LORX VERSUS RORX. In the si setting, LOR and ROR are the same up to a factor 2 in the advantage [4]. The LOR implies ROR implication is trivial and ROR implies LOR is a simple hybrid argument. We now discuss the relation between the mi counterparts, namely RORX and LORX, which is both more complex and more challenging to establish.

Theorem 2. [RORX \Rightarrow LORX] *Let $\text{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme. For all $m, t, q_c > 0$, and all vectors \mathbf{q} we have $\text{Adv}_{\text{SE},m}^{\text{lorx}}(t, \mathbf{q}, q_c) \leq 2^m \cdot \text{Adv}_{\text{SE},m}^{\text{rorx}}(t', \mathbf{q}, q_c)$, where $t' = t + \mathcal{O}(1)$. ■*

As discussed in Section 1, the multiplicative factor 2^m is often of no harm because advantages are already exponentially small in m . The factor is natural, being the mi analogue of the factor 2 appearing in the traditional si proof, and examples can be given showing that the bound is tight. The proof of the above is in [6].

The difficulty is adapting the hybrid argument technique to the mi setting. We omit the much simpler proof of the converse:

Theorem 3. [LORX \Rightarrow RORX] *Let $\text{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme. For all $m, t, q_c > 0$, and all vectors \mathbf{q} we have $\text{Adv}_{\text{SE},m}^{\text{rorx}}(t, \mathbf{q}, q_c) \leq \text{Adv}_{\text{SE},m}^{\text{lorx}}(t', \mathbf{q}, q_c)$, where $t' = t + \mathcal{O}(1)$. ■*

LORX IMPLIES AND. Intuitively, one might expect AND security to be a *stronger* requirement than LORX security, as the former seems easier to break than the latter. However we show that under a fairly minimal requirement, LORX implies AND. This brings another argument in support of LORX: Even if an application requires AND security, it turns out that proving LORX security is generally sufficient. The following theorem is to be interpreted as follows: In general, if we only know that $\text{Adv}_{\text{SE},m}^{\text{lorx}}(t, \mathbf{q}, q_c)$ is small, we do not know how to prove $\text{Adv}_{\text{SE},m}^{\text{and}}(t', \mathbf{q}, q_c)$ is also small (for $t' \approx t$), or whether this is true at all. As we sketched above, the reason is that we do not know how to use an adversary \mathcal{A} for which the $\text{AND}_{\text{SE},m}$ advantage is large to construct an adversary for which the $\text{LORX}_{\text{SE},m}$ advantage is large. Still, one would expect that such an adversary *might* more easily yield one for which the $\text{LORX}_{\text{SE},k}$ advantage is sufficiently large, for *some* $k \leq m$. The following theorem uses a probabilistic lemma due to Unger [35] to confirm this intuition.

Theorem 4. *Let $\text{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme. Further, let m, t, \mathbf{q} , and q_c be given, and assume that there exist C, ϵ , and γ such that for all $1 \leq i \leq m$,*

$$\max_{S \subseteq \{1, \dots, m\}, |S|=i} \text{Adv}_{\text{SE},i}^{\text{lorx}}(t_S^*, \mathbf{q}[S], q_c) \leq C \cdot \epsilon^i + \gamma,$$

where $\mathbf{q}[S]$ is the projection of \mathbf{q} on the components in S , and $t_S^ = t + \mathcal{O}(t_{\mathcal{E}} \cdot \sum_{i \notin S} \mathbf{q}[i])$, with $t_{\mathcal{E}}$ denoting the running time needed for one encryption with \mathcal{E} . Then, $\text{Adv}_{\text{SE},m}^{\text{and}}(t, \mathbf{q}, q_c) \leq \gamma + C \cdot \prod_{i=1}^m (1 + \epsilon_i)/2$. ■*

We are not able to prove that the converse (AND implies LORX) is true in general, but in the absence of corruptions one can upper bound $\text{Adv}_{\text{SE},m}^{\text{lorx}}(t, \mathbf{q}, 0)$ in terms of $\text{Adv}_{\text{SE},m'}^{\text{and}}(t', \mathbf{q}', 0)$ for $m' \approx 2m$ and t' and \mathbf{q}' being much larger than t, \mathbf{q} . The proof, which we omit, follows the lines of the proof of the XOR Lemma from the Direct Product Theorem given by [18], and relies on the Goldreich-Levin theorem [17]. As the loss in concrete security in this reduction is very large, and it only holds in the corruption-free case, we find this an additional argument to support the usage of the LORX metric.

3 Password-based Encryption via KDFs

We now turn to our main motivating application, that of password based encryption (PBE) as specified in PKCS#5 [32]. The schemes specified there combine a conventional mode of operation (e.g., CBC mode) with a password-based key derivation function (KDF). We start with formalizing the latter.

PASSWORD-BASED KDFS. Formally, a (k, s, c) -KDF is a deterministic map $\text{KD} : \{0, 1\}^* \times \{0, 1\}^s \rightarrow \{0, 1\}^k$ that may make use of an underlying ideal primitive. Here c is the iteration count, which specifies the multiplicative increase in work that should slow down brute force attacks.

PKCS#5 describes two KDFs [32]. We treat the first in detail and discuss the second in [6]. Let $\text{KD1}^H(pw, sa) = H^c(pw \parallel sa)$ where H^c is the function that composes H with itself c times. To generalize beyond concatenation, we can define a function $\text{Encode}(pw, sa)$ that describes how to encode its inputs onto $\{0, 1\}^*$ with efficiently computable inverse $\text{Decode}(W)$.

PBE SCHEMES. A PBE scheme is just a symmetric encryption scheme where we view the keys as passwords and key generation as a password sampling algorithm. To highlight when we are thinking of key generation as password sampling we will use \mathcal{P} to denote key generation (instead of \mathcal{K}). We will also write pw for a key that we think of as a password. Let KD be a (k, s, c) -KDF and let $\text{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an encryption scheme with \mathcal{K} outputting uniformly selected k -bit keys. Then we define the PBE scheme $\mathcal{SE}[\text{KD}, \text{SE}] = (\mathcal{P}, \bar{\mathcal{E}}, \bar{\mathcal{D}})$ as follows. Encryption $\bar{\mathcal{E}}(pw, M)$ is done via $sa \leftarrow \{0, 1\}^s$; $K \leftarrow \text{KD}(pw, sa)$; $C \leftarrow \mathcal{E}(K, M)$, returning (sa, C) as the ciphertext. Decryption recomputes the key K by reapplying the KDF and then applies \mathcal{D} . If the KDF is KD1 and the encryption scheme is CBC mode, then one obtains the first PBE scheme from PKCS#5 [32].

PASSWORD GUESSING. We aim to show that security of the above constructions holds up to the amount of work required to brute-force the passwords output by \mathcal{P} . This begs the question of how we measure the strength of a password sampler. We will formalize the hardness of guessing passwords output by some sampler \mathcal{P} via an adaptive guessing game: It challenges an adversary with guessing passwords adaptively in a setting where the attacker may, also, adaptively learn some passwords via a corruption oracle. Concretely, let $\text{GUESS}_{\mathcal{P}, m}$ be the game defined in Figure 3. A (q_t, q_c) -guessing adversary is one that makes at most q_t queries to **Test** and q_c queries to **Cor**. An adversary \mathcal{B} 's guessing advantage is $\text{Adv}_{\mathcal{P}, m}^{\text{guess}}(\mathcal{B}) = \Pr[\text{GUESS}_{\mathcal{P}, m}^{\mathcal{B}} \Rightarrow \text{true}]$. We assume without loss of generality that \mathcal{A} does not make any *pointless queries*: (1) repeated queries to **Cor** on the same value; (2) a query **Test**(i, \cdot) following a query of **Cor**(i); and (3) a query **Cor**(i) after a query **Test**(i, pw) that returned **true**. We also define a variant of the above guessing game that includes salts and allows an attacker to test password-salt pairs against all m instances simultaneously. This will be useful as an intermediate step when reducing to guessing advantage. The game $\text{saGUESS}_{\mathcal{P}, m, \rho}$ is shown in Figure 3 and we define advantage via $\text{Adv}_{\mathcal{P}, m}^{\text{sa-guess}}(\mathcal{B}) = \Pr[\text{saGUESS}_{\mathcal{P}, m}^{\mathcal{B}} \Rightarrow \text{true}]$. An easy argument proves the following lemma.

Lemma 5. *Let $m, \rho > 0$, let \mathcal{P} be a password sampler and let \mathcal{A} be an (q_t, q_c) -guessing $\text{GUESS}_{\mathcal{P}, m}$ adversary. Then there is a (q_t, q_c) -guessing $\text{saGUESS}_{\mathcal{P}, m, \rho}$ adversary \mathcal{B} such that $\text{Adv}_{\mathcal{P}, m, \rho}^{\text{sa-guess}}(\mathcal{A}) \leq \text{Adv}_{\mathcal{P}, m}^{\text{guess}}(\mathcal{B}) + m^2 \rho^2 / 2^s$. \square*

SAMPLERS WITH HIGH MIN-ENTROPY. Even though the guessing advantage precisely quantifies strength of password samplers, good upper bounds in terms of

main $\text{GUESS}_{\mathcal{P},m}$	proc. $\text{Test}(i, pw)$	proc. $\text{Cor}(i)$
$\mathbf{pw}[1], \dots, \mathbf{pw}[m] \leftarrow \mathcal{P}$	If $(pw = \mathbf{pw}[i])$ then Ret true	Ret $\mathbf{pw}[i]$
$\mathbf{pw}' \leftarrow \mathcal{B}^{\text{Test}, \text{Cor}}$	Ret \perp	
Ret $\bigwedge_{i=1}^m (\mathbf{pw}'[i] = \mathbf{pw}[i])$		
main $\text{saGUESS}_{\mathcal{P},m,\rho}$	proc. $\text{Test}(pw, sa)$	proc. $\text{Cor}(i)$
$\mathbf{pw}[1], \dots, \mathbf{pw}[m] \leftarrow \mathcal{P}$	For $i = 1$ to m do	Ret $\mathbf{pw}[i]$
For $i = 1$ to m do	For $j = 1$ to ρ do	
For $j = 1$ to ρ do	If $(pw, sa) = (\mathbf{pw}[i], \mathbf{sa}[i, j])$ then	
$\mathbf{sa}[i, j] \leftarrow \{0, 1\}^s$	Ret (i, j)	
$\mathbf{pw}' \leftarrow \mathcal{B}^{\text{Test}, \text{Cor}}(\mathbf{sa})$	Ret (\perp, \perp)	
Ret $\bigwedge_{i=1}^m (\mathbf{pw}'[i] = \mathbf{pw}[i])$		

Fig. 3. An adaptive password-guessing game.

the adversary's complexity and of some simpler relevant parameters of a password sampler are desirable. One interesting case is samplers with high min-entropy. Formally, we say that \mathcal{P} has min-entropy μ if for all pw' it holds that $\Pr[pw = pw'] \leq 2^{-\mu}$ over the coins used in choosing $pw \leftarrow \mathcal{P}$.

Theorem 6. Fix $m \geq q_c \geq 0$ and a password sampler \mathcal{P} with min-entropy μ . Let \mathcal{B} be a (q_t, q_c) -adversary for $\text{GUESS}_{\mathcal{P},m}$ making q_i queries of the form $\text{Test}(i, \cdot)$ with $q_t = q_1 + \dots + q_m$. Let $\delta = q_t/(m2^\mu)$ and let $\gamma = (m - q_c)/m$. Then $\text{Adv}_{\mathcal{P},m}^{\text{guess}}(\mathcal{B}) \leq e^{-m\Delta(\gamma,\delta)}$ where $\Delta(\gamma, \delta) = \gamma \ln(\frac{\gamma}{\delta}) + (1 - \gamma) \ln(\frac{1-\gamma}{1-\delta})$. \square

Using $\Delta(\gamma, \delta) \geq 2(\gamma - \delta)^2$, we see that to win the guessing game for q_c corruptions, $q_t \approx (m - q_c) \cdot 2^\mu$ Test queries are necessary, and the brute-force attack is optimal. Note that the above bound is the best we expect to prove: Indeed, assume for a moment that we restrict ourselves to adversaries that want to recover a subset of $m - q_c$ passwords, without corruptions, and make q_t/m queries $\text{Test}(i, \cdot)$, for each i , which are independent from queries $\text{Test}(j, \cdot)$ for other $j \neq i$. Then, each individual password is found, independently, with probability at most $q_t/(m \cdot 2^\mu)$, and if one applies the Chernoff bound, the probability that a subset of size $m - q_c$ of the passwords are retrieved is upper bounded by $e^{-m\Delta(\gamma,\delta)}$. In our case, we have additional challenges: Foremost, queries for each i are not independent. Also, the number of queries may not be the same for each index i . And finally, we allow for corruption queries.

The full proof of Theorem 6 is given in [6]. At a high level, it begins by showing how to move to a simpler setting in which the adversary wins by recovering a subset of the passwords without the aid of a corrupt oracle. The resulting setting is an example of a threshold direct product game. This allows us to apply a *generalized* Chernoff bound due to Panconesi and Srinivasan [31] (see also [20]) that reduces threshold direct product games to (non-threshold) direct product games. Finally, we apply an amplification lemma due to Maurer, Pietrzak, and Renner [25] that yields a direct product theorem for the password guessing game. Let us also note that using the same technique, the better

bound $\mathbf{Adv}_{\mathcal{P},m}^{\text{guess}}(\mathcal{B}) \leq (q_t/m2^\mu)^m$ can be proven for the special case of $(q_t, 0)$ -adversaries.

CORRELATED PASSWORDS. By taking independent samples from \mathcal{P} we have captured only the setting of independent passwords. In practice, of course, passwords may be correlated across users or, at least, user accounts. Our results extend to the setting of jointly selecting a vector of m passwords, except of course the analysis of the guessing advantage (whose proof fundamentally relies upon independence). This last only limits our ability to measure, in terms of simpler metrics like min-entropy, the difficulty of a guessing game against correlated passwords. This does not decrease the security proven, as the simulation-based paradigm we introduce below allows one to reduce to the full difficulty of the guessing game.

SIMULATION-BASED SECURITY FOR KDFs. We define an ideal-functionality style notion of security for KDFs. Figure 4 depicts two games. A message sampler \mathcal{M} is an algorithm that takes input a number r and outputs a pair of vectors $(\mathbf{pw}, \mathbf{sa})$ each having r elements and with $|\mathbf{sa}[i]| = s$ for $1 \leq i \leq r$. A simulator S is a randomized, stateful procedure. It expects oracle access to a procedure **Test** to which it can query a message. Game $\text{Real}_{\text{KD},\mathcal{M},r}$ gives a distinguisher \mathcal{D} the messages and associated derived keys. Also, \mathcal{D} can adaptively query the ideal primitive H underlying KD. Game $\text{Ideal}_{S,\mathcal{M},r}$ gives \mathcal{D} the messages and keys chosen uniformly at random. Now \mathcal{D} can adaptively query a primitive oracle implemented by a simulator S that, itself, has access to a **Test** oracle. Then we define KDF advantage by

$$\mathbf{Adv}_{\text{KD},\mathcal{M},r}^{\text{kdf}}(\mathcal{D}, S) = \Pr \left[\text{Real}_{\text{KD},\mathcal{M},r}^{\mathcal{D}} \Rightarrow 1 \right] - \Pr \left[\text{Ideal}_{S,\mathcal{M},r}^{\mathcal{D}} \Rightarrow 1 \right].$$

To be useful, we will require proving that there exists a simulator S such that for any \mathcal{D}, \mathcal{M} pair the KDF advantage is “small”.

This notion is equivalent to applying the indistinguishability framework [26] to a particular ideal KDF functionality. That functionality chooses messages according to an algorithm \mathcal{M} and outputs on its honest interface the messages and uniform keys associated to them. On the adversarial interface is the test routine which allows the simulator to learn keys associated to messages. This raises the question of why not just use indistinguishability from a RO as our target security notion. The reasons are two-fold. First, it is not clear that H^c is indistinguishable from a random oracle. Second, even if it were, a proof would seem to require a simulator that makes at least the same number of queries to the RO as it receives from the distinguisher. This rules out showing security amplification due to the iteration count c . Our approach solves both issues, since we will show KDF security for simulators that make one call to **Test** for every c made to it. For example, our simulator for KD1 will only query **Test** if a chain of c hashes leads to the being-queried point X and this chain is not a continuation of some longer chain. We formally capture this property of simulators next.

c -AMPLIFYING SIMULATORS. Let $\tau = (X_1, Y_1), \dots, (X_q, Y_q)$ be a (possibly partial) transcript of **Prim** queries and responses. We restrict attention to (k, s, c) -

main $\text{Real}_{\text{KD}, \mathcal{M}, r}$ $(\mathbf{pw}, \mathbf{sa}) \leftarrow \mathcal{M}(r)$ For $i = 1$ to r do $\mathbf{K}[i] \leftarrow \text{KD}^H(\mathbf{pw}[i], \mathbf{sa}[i])$ $b' \leftarrow \mathcal{D}^{\text{Prim}}(\mathbf{pw}, \mathbf{sa}, \mathbf{K})$ Ret b' proc. $\text{Prim}(X)$ Ret $H(X)$	main $\text{Ideal}_{S, \mathcal{M}, r}$ $(\mathbf{pw}, \mathbf{sa}) \leftarrow \mathcal{M}(r)$ For $i = 1$ to r do $\mathbf{K}[i] \leftarrow \{0, 1\}^k$ $b' \leftarrow \mathcal{D}^{\text{Prim}}(\mathbf{pw}, \mathbf{sa}, \mathbf{K})$ Ret b' proc. $\text{Prim}(X)$ Ret $S^{\text{Test}}(X)$	sub. $\text{Test}(pw, sa)$ For $i = 1$ to r do If $(\mathbf{pw}[i], \mathbf{sa}[i]) = (pw, sa)$ then Ret $\mathbf{K}[i, j]$ Ret \perp
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------

Fig. 4. Games for the simulation-based security notion for KDFs.

KDFs for which we can define a predicate $\text{final}_{\text{KD}}(X_i, \tau)$ which evaluates to true if there exists exactly one sequence of c indices $j_1 < \dots < j_c$ such that (1) $j_c = i$, (2) there exist unique (pw, sa) such that evaluating $\text{KD}^H(pw, sa)$ when H is such that $Y_j = H(X_j)$ for $1 \leq j \leq i$ results exactly in the queries X_{j_1}, \dots, X_{j_c} in any order where X_i is the last query, and (3) $\text{final}_{\text{KD}}(X_{j_r}, \tau) = \text{false}$ for all $r < c$.

Our simulators only query **Test** on queries X_i for which $\text{final}_{\text{KD}}(X_i, \tau) = \text{true}$; we call such queries *KD-completion queries* and simulators satisfying this are called *c-amplifying*. Note that (3) implies that there are at most q/c total KD-completion queries in a q -query transcript.

HASH-DEPENDENT PASSWORDS. We do not allow \mathcal{M} access to the random oracle H . This removes from consideration hash-dependent passwords. Our results should extend to cover hash-dependent passwords if one has explicit domain separation between use of H during password selection and during key derivation. Otherwise, an indistinguishability-style approach as we use here will not work due to limitations pointed out in [33]. A full analysis of the hash-dependent password setting would therefore appear to require direct analysis of PBE schemes without taking advantage of the modularity provided by simulation-based approaches.

SECURITY OF KD1. For a message sampler \mathcal{M} , let $\gamma(\mathcal{M}, r) := \Pr[\exists i \neq j : (\mathbf{pw}[i], \mathbf{sa}[i]) = (\mathbf{pw}[j], \mathbf{sa}[j])] \text{ where } (\mathbf{pw}, \mathbf{sa}) \leftarrow \mathcal{M}(r)$. We prove the following theorem in [6].

Theorem 7. *Fix $r > 0$. Let KD1 be as above. There exists a simulator S such that for all adversaries \mathcal{D} making q RO queries, of which q_c are chain completion queries, and all message samplers \mathcal{M} ,*

$$\text{Adv}_{\text{KD1}, \mathcal{M}, r}^{\text{kdf}}(\mathcal{D}, S) \leq 4\gamma(\mathcal{P}, r) + \frac{2r^2 + 7(2q + rc)^2}{2^n}.$$

*The simulator S makes at most q_c **Test** queries, and answers each query in time $O(c)$. \square*

SECURITY OF PBE. We are now in a position to analyze the security of password based encryption as used in PKCS#5. The following theorem, proved in [6], uses the multi-user left-or-right security notion from [3] whose formalization is recalled in [6]:

Theorem 8. *Let $m \geq 1$, let $\mathcal{SE}[\text{KD}, \text{SE}] = (\mathcal{P}, \overline{\mathcal{E}}, \overline{\mathcal{D}})$ be the encryption scheme built from an (k, s, c) -KDF KD and an encryption scheme $\text{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ with k -bit keys. Let \mathcal{A} be an adversary making ρ queries to $\mathbf{Enc}(i, \cdot, \cdot)$ for each $i \in \{1, \dots, m\}$ and making at most $q_c < m$ corruption queries. Let S be a c -amplifying simulator. Then there exists message sampler \mathcal{M} and adversaries \mathcal{D} , \mathcal{C} , and \mathcal{B} such that*

$$\mathbf{Adv}_{\mathcal{SE}, m}^{\text{lorx}}(\mathcal{A}) \leq m \cdot \mathbf{Adv}_{\text{SE}, \rho}^{\text{mu-lor}}(\mathcal{C}) + 2 \cdot \mathbf{Adv}_{\mathcal{P}, m, \rho}^{\text{guess}}(\mathcal{B}) + 2 \cdot \mathbf{Adv}_{\text{KD}, \mathcal{M}, m\rho}^{\text{kdf}}(\mathcal{D}, S)$$

If \mathcal{A} makes q queries to H , then: \mathcal{D} makes at most q queries to its H oracle; \mathcal{B} makes at most $\lceil q/c \rceil$ queries to \mathbf{Test} and at most q_c corruption queries; and \mathcal{C} makes a single query $\mathbf{Enc}(i, \cdot, \cdot)$ for each $1 \leq i \leq \rho$. Moreover, \mathcal{C} 's running time equals $t_{\mathcal{A}} + q \cdot t_S$ plus a small, absolute constant, and where $t_{\mathcal{A}}$ is the running time of \mathcal{A} , and t_S is the time needed by S to answer a query. Finally, $\gamma(\mathcal{M}, m\rho) \leq m^2 \rho^2 / 2^s$. \square

Note that the theorem holds even when SE is only one-time secure (meaning it can be deterministic), which implies that the analysis covers tools such as WinZip (c.f., [22]). In terms of the bound we achieve, Theorem 7 for KD1 shows that an adversary that makes $\mathbf{Adv}_{\text{KD}, \mathcal{P}^*, m\rho}^{\text{kdf}}(\mathcal{D}, S)$ large requires $q \approx 2^{n/2}$ queries to H , provided salts are large. If H is SHA-256 then this is about 2^{128} work. Likewise, a good choice of SE will ensure that $\mathbf{Adv}_{\text{SE}, \mathcal{K}, \rho}^{\text{mu-lor}}(\mathcal{C})$ will be very small. Thus the dominating term ends up the guessing advantage of \mathcal{B} against \mathcal{P} , which measures its ability to guess $m - q_c$ passwords.

Acknowledgments

Bellare was supported in part by NSF grants CCF-0915675, CNS-0904380 and CNS-1116800. Ristenpart was supported in part by NSF grant CNS-1065134. Tessaro was supported in part by NSF grants CCF-0915675, CCF-1018064.

This material is based on research sponsored by DARPA under agreement numbers FA8750-11-C-0096 and FA8750-11-2-0225. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the U.S. Government.

References

1. M. Abadi and B. Warinschi. Password-based encryption analyzed. In L. Caires, G. F. Italiano, L. Monteiro, C. Palamidessi, and M. Yung, editors, *ICALP 2005*, volume 3580 of *LNCS*, pages 664–676. Springer, July 2005.
2. O. Baudron, D. Pointcheval, and J. Stern. Extended notions of security for multicast public key cryptosystems. In U. Montanari, J. D. P. Rolim, and E. Welzl, editors, *ICALP 2000*, volume 1853 of *LNCS*, pages 499–511. Springer, July 2000.

3. M. Bellare, A. Boldyreva, and S. Micali. Public-key encryption in a multi-user setting: Security proofs and improvements. In B. Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 259–274. Springer, May 2000.
4. M. Bellare, A. Desai, E. Jorjipii, and P. Rogaway. A concrete security treatment of symmetric encryption. In *38th FOCS*, pages 394–403. IEEE Computer Society Press, Oct. 1997.
5. M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In B. Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 139–155. Springer, May 2000.
6. M. Bellare, T. Ristenpart, and S. Tessaro. Multi-instance security and its application to password-based cryptography. Cryptology ePrint Archive, Report 2012/196, 2012. <http://eprint.iacr.org/>.
7. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In V. Ashby, editor, *ACM CCS 93*, pages 62–73. ACM Press, Nov. 1993.
8. X. Boyen. Halting password puzzles: hard-to-break encryption from human-memorable keys. In *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, page 9. USENIX Association, 2007.
9. X. Boyen. New paradigms for password security (keynote lecture). In Y. Mu, W. Susilo, and J. Seberry, editors, *ACISP 08*, volume 5107 of *LNCS*, pages 1–5. Springer, July 2008.
10. R. Canetti, S. Halevi, J. Katz, Y. Lindell, and P. D. MacKenzie. Universally composable password-based key exchange. In R. Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 404–421. Springer, May 2005.
11. J.-S. Coron, Y. Dodis, C. Malinaud, and P. Puniya. Merkle-Damgård revisited: How to construct a hash function. In V. Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 430–448. Springer, Aug. 2005.
12. Y. Dodis, R. Gennaro, J. Håstad, H. Krawczyk, and T. Rabin. Randomness extraction and key derivation using the CBC, cascade and HMAC modes. In M. Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 494–510. Springer, Aug. 2004.
13. Y. Dodis, R. Impagliazzo, R. Jaiswal, and V. Kabanets. Security amplification for interactive cryptographic primitives. In O. Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 128–145. Springer, Mar. 2009.
14. R. Gennaro and Y. Lindell. A framework for password-based authenticated key exchange. In E. Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 524–543. Springer, May 2003.
15. O. Goldreich. Three XOR-Lemmas — An exposition, 1995. Available at: <http://www.wisdom.weizmann.ac.il/>.
16. O. Goldreich, R. Impagliazzo, L. A. Levin, R. Venkatesan, and D. Zuckerman. Security preserving amplification of hardness. In *31st FOCS*, pages 318–326. IEEE Computer Society Press, Oct. 1990.
17. O. Goldreich and L. A. Levin. A hard-core predicate for all one-way functions. In *21st ACM STOC*, pages 25–32. ACM Press, May 1989.
18. O. Goldreich, N. Nisan, and A. Wigderson. On Yao’s XOR-lemma. In *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation*, volume 6650 of *LNCS*, pages 273–301. Springer, 2011.
19. I. Haitner, D. Harnik, and O. Reingold. On the power of the randomized iterate. In C. Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 22–40. Springer, Aug. 2006.

20. R. Impagliazzo and V. Kabanets. Constructive proofs of concentration bounds. In M. J. Serna, R. Shaltiel, K. Jansen, and J. D. P. Rolim, editors, *APPROX-RANDOM*, volume 6302 of *LNCS*, pages 617–631. Springer, 2010.
21. J. Kelsey, B. Schneier, C. Hall, and D. Wagner. Secure applications of low-entropy keys. In E. Okamoto, G. I. Davida, and M. Mambo, editors, *1st Information Security Workshop (ISW '97)*, volume 1396 of *LNCS*, pages 121–134. Springer, Sept. 1998.
22. T. Kohno. Attacking and repairing the winZip encryption scheme. In V. Atluri, B. Pfizmann, and P. McDaniel, editors, *ACM CCS 04*, pages 72–81. ACM Press, Oct. 2004.
23. H. Krawczyk. Cryptographic extraction and key derivation: The HKDF scheme. In T. Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 631–648. Springer, Aug. 2010.
24. M. Luby and C. Rackoff. A study of password security. In C. Pomerance, editor, *CRYPTO'87*, volume 293 of *LNCS*, pages 392–397. Springer, Aug. 1988.
25. U. M. Maurer, K. Pietrzak, and R. Renner. Indistinguishability amplification. In A. Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 130–149. Springer, Aug. 2007.
26. U. M. Maurer, R. Renner, and C. Holenstein. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In M. Naor, editor, *TCC 2004*, volume 2951 of *LNCS*, pages 21–39. Springer, Feb. 2004.
27. U. M. Maurer and S. Tessaro. Computational indistinguishability amplification: Tight product theorems for system composition. In S. Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 355–373. Springer, Aug. 2009.
28. U. M. Maurer and S. Tessaro. A hardcore lemma for computational indistinguishability: Security amplification for arbitrarily weak PRGs with optimal stretch. In D. Micciancio, editor, *TCC 2010*, volume 5978 of *LNCS*, pages 237–254. Springer, Feb. 2010.
29. R. Morris and K. Thompson. Password security: a case history. *Commun. ACM*, 22:594–597, November 1979.
30. S. Myers. Efficient amplification of the security of weak pseudo-random function generators. *Journal of Cryptology*, 16(1):1–24, Jan. 2003.
31. A. Panconesi and A. Srinivasan. Randomized distributed edge coloring via an extension of the chernoff-hoeffding bounds. *SIAM J. Comput.*, 26(2):350–368, 1997.
32. PKCS #5: Password-based cryptography standard (rfc 2898). RSA Data Security, Inc., Sept. 2000. Version 2.0.
33. T. Ristenpart, H. Shacham, and T. Shrimpton. Careful with composition: Limitations of the indifferentiability framework. In K. G. Paterson, editor, *EURO-CRYPT 2011*, volume 6632 of *LNCS*, pages 487–506. Springer, May 2011.
34. S. Tessaro. Security amplification for the cascade of arbitrarily weak PRPs: Tight bounds via the interactive hardcore lemma. In Y. Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 37–54. Springer, Mar. 2011.
35. F. Unger. A probabilistic inequality with applications to threshold direct-product theorems. In *50th FOCS*, pages 221–229. IEEE Computer Society Press, Oct. 2009.
36. D. Wagner and I. Goldberg. Proofs of security for the Unix password hashing algorithm. In T. Okamoto, editor, *ASIACRYPT 2000*, volume 1976 of *LNCS*, pages 560–572. Springer, Dec. 2000.
37. A. C. Yao. Theory and applications of trapdoor functions. In *23rd FOCS*, pages 80–91. IEEE Computer Society Press, Nov. 1982.

38. F. F. Yao and Y. L. Yin. Design and analysis of password-based key derivation functions. In A. Menezes, editor, *CT-RSA 2005*, volume 3376 of *LNCS*, pages 245–261. Springer, Feb. 2005.

To Hash or Not to Hash Again? (In)differentiability Results for H^2 and HMAC

Yevgeniy Dodis¹, Thomas Ristenpart², John Steinberger³, and
Stefano Tessaro⁴

¹ New York University, dodis@cs.nyu.edu

² University of Wisconsin–Madison, rist@cs.wisc.edu

³ Tsinghua University, jpsteinb@gmail.com

⁴ Massachusetts Institute of Technology, tessaro@csail.mit.edu

Abstract. We show that the second iterate $H^2(M) = H(H(M))$ of a random oracle H cannot achieve strong security in the sense of indifferntiability from a random oracle. We do so by proving that indifferntiability for H^2 holds only with poor concrete security by providing a lower bound (via an attack) and a matching upper bound (via a proof requiring new techniques) on the complexity of any successful simulator. We then investigate HMAC when it is used as a general-purpose hash function with arbitrary keys (and not as a MAC or PRF with uniform, secret keys). We uncover that HMAC’s handling of keys gives rise to two types of weak key pairs. The first allows trivial attacks against its indifferntiability; the second gives rise to structural issues similar to that which ruled out strong indifferntiability bounds in the case of H^2 . However, such weak key pairs do not arise, as far as we know, in any deployed applications of HMAC. For example, using keys of any fixed length shorter than $d - 1$, where d is the block length in bits of the underlying hash function, completely avoids weak key pairs. We therefore conclude with a positive result: a proof that HMAC is indifferntiable from a RO (with standard, good bounds) when applications use keys of a fixed length less than $d - 1$.

Keywords: Indifferntiability, Hash functions, HMAC.

1 Introduction

Cryptographic hash functions such as those in the MD and SHA families are constructed by extending the domain of a fixed-input-length compression function via the Merkle-Damgård (MD) transform. This applies some padding to a message and then iterates the compression function over the resulting string to compute a digest value. Unfortunately, hash functions built this way are vulnerable to extension attacks that abuse the iterative structure underlying MD [22, 34]: given the hash of a message $H(M)$ an attacker can compute $H(M \parallel X)$ for some arbitrary X , even without knowing M .

In response, suggestions for shoring up the security of MD-based hash functions were made. The simplest is due to Ferguson and Schneier [20], who advocate

a hash-of-hash construction: $H^2(M) = H(H(M))$, the second iterate of H . An earlier example is HMAC [5], which similarly applies a hash function H twice, and can be interpreted as giving a hash function with an additional key input. Both constructions enjoy many desirable features: they use H as a black box, do not add large overheads, and appear to prevent the types of extension attacks that plague MD-based hash functions.

Still, the question remains whether they resist other attacks. More generally, we would like that H^2 and HMAC behave like random oracles (ROs). In this paper, we provide the first analysis of these functions as being indistinguishable from ROs in the sense of [13, 29], which (if true) would provably rule out most structure-abusing attacks. Our main results surface a seemingly paradoxical fact, that the hash-of-hash H^2 cannot be indistinguishable from a RO with good bounds, *even* if H is itself modeled as a keyed RO. We then explore the fall out, which also affects HMAC.

INDIFFERENTIABILITY. Coron et al. [13] suggest that hash functions be designed so that they “behave like” a RO. To define this, they use the indistinguishability framework of Maurer et al. [29]. Roughly, this captures that no adversary can distinguish between a pair of oracles consisting of the construction (e.g., H^2) and its underlying ideal primitive (an ideal hash H) and the pair of oracles consisting of a RO and a simulator (which is given access to the RO). A formal definition is given in Section 2. Indistinguishability is an attractive goal because of the MRH composition theorem [29]: if a scheme is secure when using a RO it is also secure when the RO is replaced by a hash construction that is indistinguishable from a RO. The MRH theorem is widely applicable (but not ubiquitously, c.f., [31]), and so showing indistinguishability provides broad security guarantees.

While there exists a large body of work showing various hash constructions to be indistinguishable from a RO (c.f., [1, 7, 11–13, 15, 16, 23]), none have yet analyzed either H^2 or HMAC. Closest is the confusingly named HMAC construction from [13], which hashes a message by computing $H^2(0^d \parallel M)$ where H is MD using a compression function with block size d bits. This is not the same as HMAC proper nor H^2 , but seems close enough to both that one would expect that the proofs of security given in [13] apply to all three.

1.1 The Second Iterate Paradox

Towards refuting the above intuition, consider that $H^2(H(M)) = H(H^2(M))$. This implies that an output of the construction $H^2(M)$ can be used as an intermediate value to compute the hash of the message $H(M)$. This property does not exist in typical indistinguishable hash constructions, which purposefully ensure that construction outputs are unlikely to coincide with intermediate values. However, and unlike where extension attacks apply (they, too, take advantage of outputs being intermediate values), there are no obvious ways to distinguish H^2 from a RO.

Our first technical contribution, then, is detailing how this structural property might give rise to vulnerabilities. Consider computing a hash chain of

length ℓ using H^2 as the hash function. That is, compute $Y = H^{2\ell}(M)$. Doing so requires 2ℓ H -applications. But the structural property of H^2 identified above means that, given M and Y one can compute $H^{2\ell}(H(M))$ using only one H -application: $H(Y) = H(H^{2\ell}(M)) = H^{2\ell}(H(M))$. Moreover, the values computed along the first hash chain, namely the values $Y_i \leftarrow H^{2i}(M)$ and $Y'_i \leftarrow H^{2i}(H(M))$ for $0 \leq i \leq \ell$ are disjoint with overwhelming probability (when ℓ is not unreasonably large). Note that for chains of RO applications, attempting to cheaply compute such a second chain would not lead to disjoint chains. This demonstrates a way in which a RO and H^2 differ.

We exhibit a cryptographic setting, called *mutual proofs of work*, in which the highlighted structure of H^2 can be exploited. In mutual proofs of work, two parties prove to each other that they have computed some asserted amount of computational effort. This task is inspired by, and similar to, client puzzles [18, 19, 24, 25, 33] and puzzle auctions [35]. We give a protocol for mutual proofs of work whose computational task is computing hash chains. This protocol is secure when using a random oracle, but when using instead H^2 an attacker can cheat by abusing the structural properties discussed above.

INDIFFERENTIABILITY LOWER BOUND. The mutual proofs of work example already points to the surprising fact that H^2 does not “behave like” a RO. In fact, it does more, ruling out proofs of indistinguishability for H^2 with good bounds. (The existence of a tight proof of indistinguishability combined with the composition theorem of [29] would imply security for mutual proofs of work, yielding a contradiction.) However, we find that the example does not surface well why simulators must fail, and the subtlety of the issues here prompt further investigation. We therefore provide a direct negative result in the form of an indistinguishability distinguisher. We prove that should the distinguisher make q_1, q_2 queries to its two oracles, then for any simulator the indistinguishability advantage of the distinguisher is lower-bounded by $1 - (q_1 q_2)/q_S - q_S^2/2^n$. (This is slightly simpler than the real bound, see Section 3.2.) What this lower bound states is that the simulator must make very close to $\min\{q_1 q_2, 2^{n/2}\}$ queries to prevent this distinguisher’s success. The result extends to structured underlying hash functions H as well, for example should H be MD-based.

To the best of our knowledge, our results are the first to show lower bounds on the number of queries an indistinguishability simulator must use. That a simulator must make a large number of queries hinders the utility of indistinguishability. When one uses the MRH composition theorem, the security of a scheme when using a monolithic RO must hold up to the number of queries the simulator makes. For example, in settings where one uses a hash function needing to be collision-resistant and attempts to conclude security via some (hypothetical) indistinguishability bound, our results indicate that the resulting security bound for the application can be at most $2^{n/4}$ instead of the expected $2^{n/2}$.

UPPER BOUNDS FOR SECOND ITERATES. We have ruled out good upper bounds on indistinguishability, but the question remains whether weak bounds exist. We provide proofs of indistinguishability for H^2 that hold up to about $2^{n/4}$ distin-

guisher queries (our lower bounds rule out doing better) when H is a RO. We provide some brief intuition about the proof. Consider an indistinguishability adversary making at most q_1, q_2 queries. The adversarial strategy of import is to compute long chains using the left oracle, and then try to “catch” the simulator in an inconsistency by querying it on a value at the end of the chain and, afterwards, filling in the intermediate values via further left and right queries. But the simulator can avoid being caught if it prepares long chains itself to help it answer queries consistently. Intuitively, as long as the simulator’s chains are a bit longer than q_1 hops, then the adversary cannot build a longer chain itself (being restricted to at most q_1 queries) and will never win. The full proofs of these results are quite involved, and so we defer more discussion until the body. We are unaware of any indistinguishability proofs that requires this kind of nuanced strategy by the simulator.

1.2 HMAC with Arbitrary Keys

HMAC was introduced by Bellare, Canetti, and Krawczyk [5] to be used as a pseudorandom function or message authentication code. It uses an underlying hash function H ; let H have block size d bits and output length n bits. Computing a hash $\text{HMAC}(K, M)$ works as follows [26]. If $|K| > d$ then redefine $K \leftarrow H(K)$. Let K' be K padded with sufficiently many zeros to get a d bit string. Then $\text{HMAC}(K, M) = H(K' \oplus \text{opad} \parallel H(K' \oplus \text{ipad} \parallel M))$ where opad and ipad are distinct d -bit constants. The original (provable security) analyses of HMAC focus on the setting that the key K is honestly generated and secret [3, 5]. But what has happened is that HMAC’s speed, ubiquity, and assumed security properties have lead it to be used in a wide variety of settings.

Of particular relevance are settings in which existing (or potential) proofs of security model HMAC as a keyed RO, a function that maps each key, message pair to an independent and uniform point. There are many examples of such settings. The HKDF scheme builds from HMAC a general-purpose key derivation function [27, 28] that uses as key a public, uniformly chosen salt. When used with a source of sufficiently high entropy, Krawczyk proves security using standard model techniques, but when not proves security assuming HMAC is a keyed RO [28]. PKCS#5 standardizes password-based key derivation functions that use HMAC with key being a (low-entropy) password [30]. Recent work provides the first proofs of security when modeling HMAC as a RO [9]. Ristenpart and Yilek [32], in the context of hedged cryptography [4], use HMAC in a setting whose cryptographic security models allow adversarially specified keys. Again, proofs model HMAC as a keyed RO.

As mentioned previously, we would expect a priori that one can show that HMAC is indistinguishable from a keyed RO even when the attacker can query arbitrary keys. Then one could apply the composition theorem of [29] to derive proofs of security for the settings just discussed.

WEAK KEY PAIRS IN HMAC. We are the first to observe that HMAC has weak key pairs. First, there exist $K \neq K'$ for which $\text{HMAC}(K, M) = \text{HMAC}(K', M)$.

These pairs of keys arise because of HMAC’s ambiguous encoding of differing-length keys. Trivial examples of such “colliding” keys include any K, K' for which either $|K| < d$ and $K' = K \parallel 0^s$ (for any $1 \leq s \leq d - |K|$), or $|K| > d$ and $K' = H(K)$. Colliding keys enable an easy attack that distinguishes $\text{HMAC}(\cdot, \cdot)$ from a random function $\mathcal{R}(\cdot, \cdot)$, which also violates the indistinguishability of HMAC. On the other hand, as long as H is collision-resistant, two keys of the same length can never collide. Still, even if we restrict attention to (non-colliding) keys of a fixed length, there still exist weak key pairs, but of a different form that we term ambiguous. An example of an ambiguous key pair is K, K' of length d bits such that $K \oplus \text{ipad} = K' \oplus \text{opad}$. Because the second least significant bit of ipad and opad differ (see Section 4) and assuming $d > n - 2$, ambiguous key pairs of a fixed length k only exist for $k \in \{d - 1, d\}$. The existence of ambiguous key pairs in HMAC leads to negative results like those given for H^2 . In particular, we straightforwardly extend the H^2 distinguisher to give one that lower bounds the number of queries any indistinguishability simulator must make for HMAC.

UPPER BOUNDS FOR HMAC. Fortunately, it would seem that weak key pairs do not arise in typical applications. Using HMAC with keys of some fixed bit length smaller than $d - 1$ avoids weak key pairs. This holds for several applications, for example the recommendation with HKDF is to use n -bit uniformly chosen salts as HMAC keys. This motivates finding positive results for HMAC when one avoids the corner cases that allow attackers to exploit weak key pairs.

Indeed, as our main positive result, we prove that, should H be a RO or an MD hash with ideal compression functions, *HMAC is indistinguishable from a keyed RO for all distinguishers that do not query weak key pairs*. Our result holds for the case that the keys queried are of length d or less. This upper bound enjoys the best, birthday-bound level of concrete security possible (up to small constants), and provides the *first positive result about the indistinguishability of the HMAC construction*.

1.3 Discussion

The structural properties within H^2 and HMAC are, in theory, straightforward to avoid. Indeed, as mentioned above, Coron et al. [13] prove indistinguishability from a RO the construction $H^2(0^d \parallel M)$ where H is MD using a compression function with block size d bits and chaining value length $n \leq d$ bits. Analogously, our positive results about HMAC imply as a special case that $\text{HMAC}(K, M)$, for any fixed constant K , is indistinguishable from a RO.

We emphasize that we are unaware of any deployed cryptographic application for which the use of H^2 or HMAC leads to a vulnerability. Still, our results show that future applications should, in particular, be careful when using HMAC with keys which are under partial control of the attacker. More importantly, our results demonstrate the importance of provable security in the design of hash functions (and elsewhere in cryptography), as opposed to the more common “attack-fix” cycle. For example, the hash-of-hash suggestion of Ferguson and Schneier [20] was motivated by preventing the extension attack. Unfortunately,

in so doing they accidentally introduced a more subtle (although less dangerous) attack, which was not present on the original design.⁵ Indeed, we discovered the subtlety of the problems within H^2 and HMAC, including our explicit attacks, only after attempting to prove indistinguishability of these constructions (with typical, good bounds). In contrast, the existing indistinguishability proofs of (seemingly) small modifications of these hash functions, such as $H^2(0^d \parallel M)$ [13], provably rule out these attacks.

1.4 Prior Work

There exists a large body of work showing hash functions are indistinguishable from a RO (c.f., [1, 7, 11–13, 15, 16, 23]), including analyses of variants of H^2 and HMAC. As mentioned, a construction called HMAC was analyzed in [13] but this construction is not HMAC as standardized. Krawczyk [28] suggests that the analysis of $H^2(0 \parallel M)$ extends to the case of HMAC, but does not offer proof.⁶ HMAC has received much analysis in other contexts. Proofs of its security as a pseudorandom function under reasonable assumptions appear in [3, 5]. These rely on keys being uniform and secret, making the analyses inapplicable for other settings. Analysis of HMAC’s security as a randomness extractor appear in [14, 21]. These results provide strong information theoretic guarantees that HMAC can be used as a key derivation function, but only in settings where the source has a relatively large amount of min-entropy. This requirement makes the analyses insufficient to argue security in many settings of practical importance. See [28] for further discussion.

FULL VERSION. Due to space constraints, many of our technical results and proofs are deferred to the full version of this paper [17].

2 Preliminaries

NOTATION AND GAMES. We denote the empty string by λ . If $|X| < |Y|$ then $X \oplus Y$ signifies that the X is padded with $|Y| - |X|$ zeros first. For set \mathcal{X} and value x , we write $\mathcal{X} \stackrel{\cup}{\leftarrow} x$ to denote $\mathcal{X} \leftarrow \mathcal{X} \cup \{x\}$. For non-empty sets $Keys$, Dom , and Rng with $|Rng|$ finite, a random oracle $f: Keys \times Dom \rightarrow Rng$ is a function taken randomly from the space of all possible functions $Keys \times Dom \rightarrow Rng$. We will sometimes refer to random oracles as keyed when $Keys$ is non-empty, whereas we omit the first parameter when $Keys = \emptyset$.

We use code-based games [10] to formalize security notions and within our proofs. In the execution of a game G with adversary \mathcal{A} , we denote by $G^{\mathcal{A}}$ the event that the game outputs true and by $\mathcal{A}^G \Rightarrow y$ the event that the adversary

⁵ We note the prescience of the proposers of H^2 , who themselves suggested further analysis was needed [20].

⁶ Fortunately, the HKDF application of [28] seems to avoid weak key pairs, and thus our positive results for HMAC appear to validate this claim [28] for this particular application.

outputs y . Fixing some RAM model of computation, our convention is that the running time $\text{Time}(\mathcal{A})$ of an algorithm \mathcal{A} includes its code size. Queries are unit cost, and we will restrict attention to the absolute worst case running time which must hold regardless of queries are answered.

HASH FUNCTIONS. A *hash function* $H[P]: \text{Keys} \times \text{Dom} \rightarrow \text{Rng}$ is a family of functions from Dom to Rng , indexed by a set Keys , that possibly uses (black-box) access to an underlying primitive P (e.g., a compression function). We call the hash function *keyed* if Keys is non-empty, and *key-less* otherwise. (In the latter case, we omit the first parameter.) We assume that the number of applications of P in computing $H[P](K, M)$ is the same for all K, M with the same value of $|K| + |M|$. This allows us to define the *cost* of computing a hash function $H[P]$ on a key and message whose combined length is ℓ , denoted $\text{Cost}(H, \ell)$, as the number of calls to P required to compute $H[P](K, M)$ for K, M with $|K| + |M| = \ell$. For a keyed random oracle $\mathcal{R}: \text{Keys} \times \text{Dom} \rightarrow \text{Rng}$, we fix the convention that $\text{Cost}(\mathcal{R}, \ell) = 1$ for any ℓ for which there exists a key $K \in \text{Keys}$ and message $M \in \text{Dom}$ such that $|K| + |M| = \ell$.

A *compression function* is a hash function for which $\text{Dom} = \{0, 1\}^n \times \{0, 1\}^d$ and $\text{Rng} = \{0, 1\}^n$ for some numbers $n, d > 0$. Our focus will be on keyless compression functions, meaning those of the form $f: \{0, 1\}^n \times \{0, 1\}^d \rightarrow \{0, 1\}^n$. Our results lift in a straightforward way to the dedicated-key setting [8]. The ℓ -th iterate of $H[P]$ is denoted $H^\ell[P]$, and defined for $\ell > 0$ by $H^\ell[P](X) = H[P](H[P](\dots H[P](X))\dots)$ where the number of applications of H is ℓ . We let $H^0[P](X) = X$. We will often write H instead of $H[P]$ when the underlying primitive P is clear or unimportant.

MERKLE-DAMGÅRD. Let $\text{Pad}: \{0, 1\}^{\leq L} \rightarrow (\{0, 1\}^n)^+$ be an injective padding function. The one used in many of the hash functions within the SHA family outputs $M \parallel 10^r \parallel \langle |M| \rangle_{64}$ where $\langle |x| \rangle_{64}$ is the encoding of the length of M as a 64-bit string and r is the smallest number making the length a multiple of d . This makes $L = 2^{64} - 1$. The function $\text{MD}[f]: (\{0, 1\}^n)^+ \rightarrow \{0, 1\}^n$ is defined as

$$\text{MD}[f](M) = f(f(\dots f(f(\text{IV}, M_1), M_2), \dots), M_k)$$

where $|M| = kd$ and $M_1 \parallel \dots \parallel M_k$. The function $\text{SMD}[f]: \{0, 1\}^{\leq L} \rightarrow \{0, 1\}^n$ is defined as $\text{SMD}[f](M) = \text{MD}[f](\text{Pad}(M))$.

INDIFFERENTIABILITY FROM A RO. Let $\mathcal{R}: \text{Keys} \times \text{Dom} \rightarrow \text{Rng}$ be a random oracle. Consider a hash construction $H[P]: \text{Keys} \times \text{Dom} \rightarrow \text{Rng}$ from an ideal primitive P . Let game $\text{Real}_{H[P]}$ be the game whose main procedure runs an adversary $\mathcal{A}^{\text{Func, Prim}}$ and returns the bit that \mathcal{A} outputs. The procedure **Func** on input $K \in \text{Keys}$ and $M \in \text{Dom}$ returns $H[P](K, M)$. The procedure **Prim** on input X returns $P(X)$. For a simulator \mathcal{S} , let game $\text{Ideal}_{\mathcal{R}, \mathcal{S}}$ be the game whose main procedure runs an adversary $\mathcal{A}^{\text{Func, Prim}}$ and returns the bit that \mathcal{A} outputs. The procedure **Func** on input $K \in \text{Keys}$ and $M \in \text{Dom}$ returns $\mathcal{R}(K, M)$. The procedure **Prim** on input X returns $\mathcal{S}^{\mathcal{R}}(X)$. The indistinguishability advantage

of \mathcal{D} is defined as

$$\mathbf{Adv}_{H[P], \mathcal{R}, \mathcal{S}}^{\text{indiff}}(\mathcal{D}) = \Pr \left[\text{Real}_{H[P]}^{\mathcal{D}} \Rightarrow y \right] - \Pr \left[\text{Ideal}_{\mathcal{R}, \mathcal{S}}^{\mathcal{D}} \Rightarrow y \right] .$$

We focus on simulators that must work for any adversary, though our negative results extend as well to the weaker setting in which the simulator can depend on the adversary. The total query cost σ of an adversary \mathcal{D} is the cumulative cost of all its `Func` queries plus q_2 . (This makes σ the total number of P uses in game $\text{Real}_{H[P]}^{\mathcal{D}}$. In line with our worst-case conventions, this means the same maximums hold in $\text{Ideal}_{\mathcal{R}, \mathcal{S}}^{\mathcal{D}}$ although here it does not translate to P applications.)

We note that when `Keys` is non-empty, indistinguishability here follows [8] and allows the distinguisher to choose keys during an attack. This reflects the desire for a keyed hash function to be indistinguishable from a keyed random oracle for arbitrary uses of the key input.

3 Second Iterates and their Security

Our investigation begins with the second iterate of a hash function, meaning $H^2(M) = H(H(M))$ where $H: \text{Dom} \rightarrow \text{Rng}$ for sets $\text{Dom} \supseteq \text{Rng}$. For simplicity, let $\text{Rng} = \{0, 1\}^n$ and assume that H is itself modeled as a RO. Is H^2 good in the sense of being like a RO? Given that we are modeling H as a RO, we would expect that the answer would be “yes”. The truth is more involved. As we’ll see in Section 4, similar subtleties exist in the case of the related HMAC construction.

We start with the following observations. When computing $H^2(M)$ for some M , we refer to the value $H(M)$ as an intermediate value. Then, we note that the value $Y = H^2(M)$ is in fact the intermediate value used when computing $H^2(X)$ for $X = H(M)$. Given $Y = H^2(M)$, then, one can compute $H^2(H(M))$ directly by computing $H(Y)$. That the hash value Y is also the intermediate value used in computing the hash of another message is cause for concern: other hash function constructions that are indistinguishable from a RO (c.f., [2, 7, 8, 13, 23]) explicitly attempt to ensure that outputs are *not* intermediate values (with overwhelming probability over the randomness of the underlying idealized primitive). Moreover, prior constructions for which hash values are intermediate values have been shown to *not* be indistinguishable from a RO. For example Merkle-Damgård-based iterative hashes fall to extension attacks [13] for this reason. Unlike with Merkle-Damgård, however, it is not immediately clear how an attacker might abuse the structure of H^2 .

We turn our attention to hash chains, where potential issues arise. For a hash function H , we define a hash chain $Y = (Y_0, \dots, Y_\ell)$ to be a sequence of $\ell + 1$ values where Y_0 is a message and $Y_i = H(Y_{i-1})$ for $1 \leq i \leq \ell$. Likewise when using H^2 a hash chain $Y = (Y_0, \dots, Y_\ell)$ is a sequence of $\ell + 1$ values where Y_0 is a message and $Y_i = H^2(Y_{i-1})$ for $1 \leq i \leq \ell$. We refer to Y_0 as the *start* of the hash chain and Y_ℓ as the *end*. Two chains Y, Y' are *non-overlapping* if no value in one chain occurs in the other, meaning $Y_i \neq Y'_j$ for all $0 \leq i \leq \ell$.

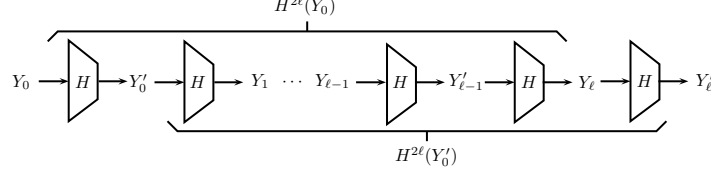


Fig. 1. Diagram of two hash chains $Y = (Y_0, \dots, Y_\ell)$ and $Y' = (Y'_0, \dots, Y'_\ell)$ for hash function H^2 .

For any hash function and given the start and end of a hash chain $Y = (Y_0, \dots, Y_\ell)$, one can readily compute the start and end of a new chain with just two hash calculations. That is, set $Y'_0 \leftarrow H(Y_0)$ and $Y'_\ell \leftarrow H(Y_\ell)$. However, the chain $Y' = (Y'_0, \dots, Y'_\ell)$ and the chain Y overlap. For good hash functions (i.e., ones that behave like a RO) computing the start and end of a non-overlapping chain given the start and end of a chain Y_0, Y_ℓ requires at least ℓ hash computations (assuming $\ell \ll 2^{n/2}$).

Now consider H^2 . Given the start and end of a chain $Y = (Y_0, \dots, Y_\ell)$, one can readily compute a non-overlapping chain $Y' = (Y'_0, \dots, Y'_\ell)$ using just two hash computations instead of the expected 2ℓ computations. Namely, let $Y'_0 \leftarrow H(Y_0)$ and $Y'_\ell \leftarrow H(Y_\ell)$. Then these are the start and end of the chain $Y' = (Y'_0, \dots, Y'_\ell)$ because

$$H^{2\ell}(Y'_0) = H^{2\ell}(H(Y_0)) = H(H^{2\ell}(Y_0))$$

which we call the chain-shift property of H^2 . Moreover, assuming H is itself a RO outputting n -bit strings, the two chains Y, Y' do not overlap with probability at least $1 - (2\ell + 2)^2/2^n$. Figure 1 provides a pictorial diagram of the two chains Y and Y' .

3.1 A Vulnerable Application: Mutual Proofs of Work

In the last section we saw that the second iterate fails to behave like a RO in the context of hash chains. But the security game detailed in the last section may seem far removed from real protocols. For example, it's not clear where an attacker would be tasked with computing hash chains in a setting where it, too, was given an example hash chain. We suggest that just such a setting could arise in protocols in which parties want to assert to each other, in a verifiable way, that they performed some amount of computation. Such a setting could arise when parties must (provably) compare assertions of computational power, as when using cryptographic puzzles [18, 19, 24, 25, 33, 35]. Or this might work when trying to verifiably calibrate differing computational speeds of the two parties' computers. We refer to this task as a *mutual proof of work*.

MUTUAL PROOFS-OF-WORK. For the sake of brevity, we present an example hash-chain-based protocol and dispense with a more general treatment of mutual proofs of work. Consider the two-party protocol shown in the left diagram

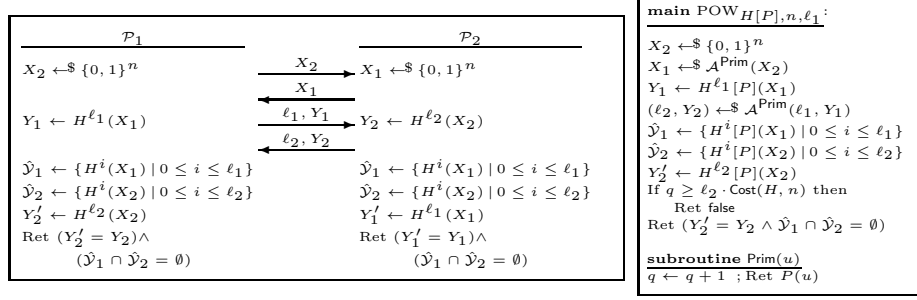


Fig. 2. Example protocol (left) and adversarial \mathcal{P}_2 security game (right) for mutual proofs of work.

of Figure 2. Each party initially chooses a random nonce and sends it to the other. Then, each party computes a hash chain of some length—chosen by the computing party—starting with the nonce chosen by the other party, and sends the chain’s output along with the chain’s length to the other party. At this point, both parties have given a witness that they performed a certain amount of work. So now, each party checks the other’s asserted computation, determining if the received value is the value resulting from chaining together the indicated number of hash applications and checking that the hash chains used by each party are non-overlapping. Note that unlike puzzles, which require fast verification, here the verification step is as costly as puzzle solution.

The goal of the protocol is to ensure that the other party did compute exactly their declared number of iterations. Slight changes to the protocol would lead to easy ways of cheating. For example, if during verification the parties did not check that the chains are non-overlapping, then \mathcal{P}_2 can easily cheat by choosing X_1 so that it can reuse a portion of the chain computed by \mathcal{P}_1

Security would be achieved should no cheating party succeed at convincing an honest party using less than ℓ_1 (resp. ℓ_2) work to compute Y_1 (resp. Y_2). The game POW $_{H[P],n,\ell_1}$ formalizes this security goal for a cheating \mathcal{P}_2 ; see the right portion of Figure 2. We let $\text{Adv}_{H[P],n,\ell_1}^{\text{pow}}(\mathcal{A}) = \Pr \left[\text{POW}_{H[P],n,\ell_1}^{\mathcal{A}} \right]$. Note that the adversary \mathcal{A} only wins should it make $q < \ell_2 \cdot \text{Cost}(H, n)$ queries, where ℓ_2 is the value it declared and $\text{Cost}(H)$ is the cost of computing H . Again we will consider both the hash function $H[P](M) = P(M)$ that just applies a RO P and also $H^2[P](M) = P(P(M))$, the second iterate of a RO. In the former case the can make only $\ell_2 - 1$ queries and in the latter case $2\ell_2 - 1$.

When $H[P](M) = P(M)$, no adversary making $q < \ell_2$ queries to Prim can win the POW $_{H[P],n,\ell_1}$ game with high advantage. Intuitively, the reason is that, despite being given X_1 and Y_1 where $Y_1 = P^{\ell_1}(X_1)$, a successful attacker must still compute a full ℓ_2 -length chain and this requires ℓ_2 calls to P . A more formal treatment appears in the full version.

ATTACK AGAINST ANY SECOND ITERATE. Now let us analyze this protocol's security when we use as hash function $H^2[P] = P(P(M))$ for a RO $P: \text{Dom} \rightarrow \text{Rng}$ with $\text{Rng} \subseteq \text{Dom}$. We can abuse the chain-shift property of H^2 in order to win the $\text{POW}_{H^2, P, n, \ell_1}$ game for any $n > 0$ and $\ell_1 > 2$. Our adversary \mathcal{A} works as follows. It receives X_2 and then chooses its nonce as $X_1 \leftarrow \text{Prim}(X_2)$. When it later receives $Y_1 = P^{2\ell_1}(X_1)$, the adversary proceeds by setting $\ell_2 = \ell_1 + 1$ and setting $Y_2 \leftarrow \text{Prim}(Y_1)$. Then by the chain-shift property we have that

$$Y_2 = P(Y_1) = P(P^{2\ell_1}(X_1)) = P(P^{2\ell_1}(P(X_2))) = P^{2\ell_2}(X_2) .$$

The two chains will be non-overlapping with high probability (over the coins used by P). Finally, \mathcal{A} makes only 2 queries to Prim , so the requirement that $q < 2\ell_2$ is met whenever $\ell_1 > 1$.

DISCUSSION. As far as we are aware, mutual proofs of work have not before been considered — the concept may indeed be of independent interest. A full treatment is beyond the scope of this work. We also note that, of course, it is easy to modify the protocols using H^2 to be secure. Providing secure constructions was not our goal, rather we wanted to show protocols which are insecure using H^2 but secure when H^2 is replaced by a monolithic RO. This illustrates how, hypothetically, the structure of H^2 could give rise to subtle vulnerabilities in an application.

3.2 Indifferentiability Lower and Upper Bounds

In this section we prove that *any* indifferentiability proof for the double iterate H^2 is subject to inherent quantitative limitations. Recall that indifferentiability asks for a simulator \mathcal{S} such that no adversary can distinguish between the pair of oracles $H^2[P], P$ and \mathcal{R}, \mathcal{S} where P is some underlying ideal primitive and \mathcal{R} is a RO with the same domain and range as H^2 . The simulator can make queries to \mathcal{R} to help it in its simulation of P . Concretely, building on the ideas behind the above attacks in the context of hash chains, we show that in order to withstand a differentiating attack with q queries, any simulator for $H^2[P]$, for *any* hash construction $H[P]$ with output length n , must issue *at least* $\Omega(\min\{q^2, 2^{n/2}\})$ queries to the RO \mathcal{R} . As we explain below, such a lower bound severely limits the concrete security level which can be inferred by using the composition theorem for indifferentiability, effectively neutralizing the benefits of using indifferentiability in the first place.

THE DISTINGUISHER. In the following, we let $H = H[P]$ be an *arbitrary* hash function with n -bit outputs relying on a primitive P , such as a fixed input-length random oracle or an ideal cipher. We are therefore addressing an arbitrary second iterate, and not focusing on some particular ideal primitive P (such as a RO as in previous sections) or construction H . Indeed, H could equally well be Merkle-Damgård and P an ideal compression function, or H could be any number of indifferentiable hash constructions using appropriate ideal primitive P .

Recall that Func and Prim are the oracles associated with construction and primitive queries to $H^2 = H^2[P]$ and P , respectively. Let w, ℓ be parameters (for

now, think for convenience of $w = \ell$). The attacker $\mathcal{D}_{w,\ell}$ starts by issuing ℓ queries to **Func** to compute a *chain* of n -bit values $(x_0, x_1, \dots, x_\ell)$ where $x_i = H^2(x_{i-1})$ and x_0 is a random n -bit string. Then, it also picks a random index $j \in [1..w]$, and creates a list of n -bit strings $\mathbf{u}[1], \dots, \mathbf{u}[w]$ with $\mathbf{u}[j] = x_\ell$, and all remaining $\mathbf{u}[i]$ for $i \neq j$ are chosen uniformly and independently. Then, for all $i \in [1..w]$, the distinguisher $\mathcal{D}_{w,\ell}$ proceeds in asking all **Prim** queries in order to compute $\mathbf{v}[i] = H(\mathbf{u}[i])$. Subsequently, the attacker compute $y_0 = H(x_0)$ via **Prim** queries, and also computes the chain $(y_0, y_1, \dots, y_\ell)$ such that $y_i = H^2(y_{i-1})$ by making ℓ **Func** queries. Finally, it decides to output 1 if and only if $y_\ell = \mathbf{v}[j]$ and x_ℓ as well as $\mathbf{v}[i]$ for $i \neq j$ are not in $\{y_0, y_1, \dots, y_\ell\}$. The attacker $\mathcal{D}_{w,\ell}$ therefore issues a total of 2ℓ **Func** queries and $(2w + 1) \cdot \text{Cost}(H, n)$ **Prim** queries.

In the real-world experiment, the distinguisher $\mathcal{D}_{w,\ell}$ outputs 1 with very high probability, as the condition $y_\ell = \mathbf{v}[j]$ *always* holds by the chain-shifting property of H^2 . In fact, the only reason for \mathcal{D} outputting 0 is that one of x_ℓ and $\mathbf{v}[i]$ for $i \neq j$ incidentally happens to be in $\{y_0, y_1, \dots, y_\ell\}$. The (typically small) probability that this occurs obviously depends on the particular construction $H[P]$ at hand; it is thus convenient to define the shorthand

$$p(H, w, \ell) = \Pr[\{x_\ell, H(U_1), \dots, H(U_{w-1})\} \cap \{y_0, y_1, \dots, y_\ell\} \neq \emptyset],$$

where $x_0, y_0, x_1, \dots, y_{\ell-1}, x_\ell, y_\ell$ are the intermediate value of a chain of $2\ell + 1$ consecutive evaluations of $H[P]$ starting at a random n -bit string x_0 , and U_1, \dots, U_{w-1} are further independent random n -bit values. In the full version of this paper we prove that for $H[P] = P = \mathcal{R}$ for a random oracle $\mathcal{R} : \{0, 1\}^* \rightarrow \{0, 1\}^n$ we have $p(H, w, \ell) = \Theta((w\ell + \ell^2)/2^n)$. Similar reasoning can be applied to essentially all relevant constructions.

In contrast, in the ideal-world experiment, we expect the simulator to be completely ignorant about the choice of j as long as it does not learn x_0 , and in particular it does not know j while answering the **Prim** queries associated with the evaluations of $H(\mathbf{u}[i])$. Consequently, the condition required for $\mathcal{D}_{w,\ell}$ to output 1 appears to force the simulator, for all $i \in [1..w]$, to prepare a distinct chain of ℓ consecutive \mathcal{R} evaluations ending in $\mathbf{v}[i]$, hence requiring $w \cdot \ell$ random oracle queries.

The following theorem quantifies the advantage achieved by the above distinguisher $\mathcal{D}_{w,\ell}$ in differentiating against any simulator for the construction $H[P]$. Its proof is given in the full version.

Theorem 1. [Attack against H^2] *Let $H[P]$ be an arbitrary hash construction with n -bit outputs, calling a primitive P , and let $\mathcal{R} : \{0, 1\}^* \rightarrow \{0, 1\}^n$ be a random oracle. For all integer parameters $w, \ell \geq 1$, there exists an adversary $\mathcal{D}_{w,\ell}$ making 2ℓ **Func**-queries and $(w + 1) \cdot \text{Cost}(H, n)$ **Prim**-queries such that for all simulators \mathcal{S} ,*

$$\text{Adv}_{H^2[P], \mathcal{R}, \mathcal{S}}^{\text{indiff}}(\mathcal{D}_{w,\ell}) \geq 1 - p(H, w, \ell) - \frac{5\ell^2}{2^{n+1}} - \frac{q_S \ell}{2^n} - \frac{q_S^2}{2^n} - \frac{q_S}{w \cdot \ell} - \frac{1}{w},$$

where q_S is the overall number of \mathcal{R} queries by \mathcal{S} when replying to $\mathcal{D}_{w,\ell}$'s **Prim** queries. ■

DISCUSSION. We now elaborate on Theorem 1. If we consider the distinguisher $\mathcal{D}_{w,\ell}$ from Theorem 1, we observe that by the advantage lower bound in the theorem statement, if $\ell, w \ll 2^{n/4}$ and consequently $p(H, w, \ell) \approx 0$, the number of queries made by the simulator, denoted $q_S = q_S(2\ell, w + 1)$ must satisfy $q_S = \Omega(w \cdot \ell) = \Omega(q_1 \cdot q_2)$ to ensure a sufficiently small indistinguishability advantage. This in particular means that in the case where both q_1 and q_2 are large, the simulator must make a *quadratic* effort to prevent the attacker from distinguishing. Below, in Theorem 2, we show that this simulation effort is essentially optimal.

In many scenarios, this quadratic lower bound happens to be a problem, as we now illustrate. As a concrete example, let $\mathcal{SS} = (\text{key}, \text{sign}, \text{ver})$ be an arbitrary signature scheme signing n bits messages, and let $\widetilde{\mathcal{SS}}[\mathcal{R}] = (\widetilde{\text{key}}^{\mathcal{R}}, \widetilde{\text{sign}}^{\mathcal{R}}, \widetilde{\text{ver}}^{\mathcal{R}})$ for $\mathcal{R} : \{0, 1\}^* \rightarrow \{0, 1\}^n$ be the scheme obtained via the hash-then-sign paradigm such that $\widetilde{\text{sign}}^{\mathcal{R}}(sk, m) = \text{sign}(sk, \mathcal{R}(m))$. It is well known that for an adversary \mathcal{B} making q_{sign} signing and $q_{\mathcal{R}}$ random oracle queries, there exists an adversary \mathcal{C} making q_{sign} signing queries such that

$$\mathbf{Adv}_{\widetilde{\mathcal{SS}}[\mathcal{R}]}^{\text{uf-cma}}(\mathcal{B}^{\mathcal{R}}) \leq \frac{(q_{\text{sign}} + q_{\mathcal{R}})^2}{2^n} + \mathbf{Adv}_{\mathcal{SS}}^{\text{uf-cma}}(\mathcal{C}), \quad (1)$$

where $\mathbf{Adv}_{\widetilde{\mathcal{SS}}[\mathcal{R}]}^{\text{uf-cma}}(\mathcal{B}^{\mathcal{R}})$ and $\mathbf{Adv}_{\mathcal{SS}}^{\text{uf-cma}}(\mathcal{C})$ denote the respective advantages in the standard uf-cma game for security of signature schemes (with and without a random oracle, respectively). This in particular means that $\widetilde{\mathcal{SS}}$ is secure for q_{sign} and $q_{\mathcal{R}}$ as large as $\Theta(2^{n/2})$, provided \mathcal{SS} is secure for q_{sign} signing queries. However, let us now replace \mathcal{R} by $H^2[P]$ for an arbitrary construction $H = H[P]$. Then, for all adversaries \mathcal{A} making q_P queries to P and q_{sign} signing queries, we can combine the concrete version of the MRH composition theorem proven in [31] and (1) to infer that there exists an adversary \mathcal{C} and a distinguisher \mathcal{D} such that

$$\mathbf{Adv}_{\widetilde{\mathcal{SS}}[H^2[P]]}^{\text{uf-cma}}(\mathcal{A}^P) \leq \Theta\left(\frac{(q_{\text{sign}} \cdot q_P)^2}{2^n}\right) + \mathbf{Adv}_{\mathcal{SS}}^{\text{uf-cma}}(\mathcal{C}) + \mathbf{Adv}_{H^2[P], \mathcal{R}, \mathcal{S}}^{\text{indiff}}(\mathcal{D}),$$

where \mathcal{C} makes q_{sign} signing queries. Note that even if the term $\mathbf{Adv}_{H^2[P], \mathcal{R}, \mathcal{S}}^{\text{indiff}}(\mathcal{D})$ is really small, this new bound can only ensure security for the resulting signature scheme as long as $q_{\text{sign}} \cdot q_P = \Theta(2^{n/2})$, i.e., if $q_{\text{sign}} = q_P$, we only get security up to $\Theta(2^{n/4})$ queries, a remarkable loss with respect to the security bound in the random oracle model.

We note that of course this does *not* mean that $H^2[P]$ for a concrete H and P is unsuitable for a certain application, such as hash-then-sign. In fact, $H^2[P]$ may well be optimally collision resistant. However, our result shows that a sufficiently strong security level cannot be inferred from *any* indistinguishability statement via the composition theorem, taking us back to a direct ad-hoc analysis and completely loosing the one main advantage of having indistinguishability in the first place.

UPPER BOUND. Our negative results do not rule out positive results completely: there could be indistinguishability upper bounds, though for simulators that make

around $\mathcal{O}(q^2)$ queries. Ideally, we would like upper bounds that match closely the lower bounds given in prior sections. We do so for the special case of $H^2[g](M) = g(g(M))$ for $g: \{0, 1\}^n \rightarrow \{0, 1\}^n$ being a RO.

Theorem 2. *Let $q_1, q_2 \geq 0$ and $N = 2^n$. Let $g: \{0, 1\}^n \rightarrow \{0, 1\}^n$ and $\mathcal{R}: \{0, 1\}^n \rightarrow \{0, 1\}^n$ be uniform random functions. Then there exists a simulator \mathcal{S} such that*

$$\text{Adv}_{G[g], \mathcal{R}, \mathcal{S}}^{\text{indiff}}(\mathcal{D}) \leq \frac{2((4q_1 + 3)q_2 + 2q_1)^2}{N} + \frac{2((4q_1 + 3)q_2 + 2q_1)(q_1 + q_2)}{(N - 2q_2 - 2q_1)}$$

for any adversary \mathcal{D} making at most q_1 queries to its left oracle and at most q_2 queries to its right oracle. Moreover, for each query answer that it computes, \mathcal{S} makes at most $3q_1 + 1$ queries to RO and runs in time $\mathcal{O}(q_1)$. \square

The proof of the theorem appears in the full version of the paper. We note that the simulator used must know the maximum number of queries the attacker will make, but does not otherwise depend on the adversary's strategy. The security bound of the theorem is approximately $(q_1 q_2)^2 / N$, implying that security holds up to $q_1 q_2 \approx 2^{n/2}$.

4 HMAC as a General-purpose Keyed Hash Function

HMAC [5] uses a hash function to build a keyed hash function, i.e. one that takes both a key and message as input. Fix some hash function⁷ $H: \{0, 1\}^* \rightarrow \{0, 1\}^n$. HMAC assumes this function H is built by iterating an underlying compression function with a message block size of $d \geq n$ bits. We define the following functions:

$$\begin{aligned} F_K(M) &= H((\rho(K) \oplus \text{ipad}) \parallel M) \\ G_K(M) &= H((\rho(K) \oplus \text{opad}) \parallel M) \end{aligned} \quad \text{where } \rho(K) = \begin{cases} H(K) & \text{if } |K| > d \\ K & \text{otherwise.} \end{cases}$$

The two constants used are $\text{ipad} = 0x36^{d/8}$ and $\text{opad} = 0x5c^{d/8}$. These constants are given in hexadecimal, translating to binary gives $0x36 = 0011\,0110_2$ and $0x5c = 0101\,1100_2$. Recall that we have defined the \oplus operator so that, if $|K| < d$, it first silently pads out the shorter string by sufficiently many zeros before computing the bitwise xor. It will also be convenient to define $\text{xpad} = \text{ipad} \oplus \text{opad}$. The function $\text{HMAC}: \{0, 1\}^* \times \{0, 1\}^* \rightarrow \{0, 1\}^n$ is defined by

$$\text{HMAC}(K, M) = G_K(F_K(M)) = (G_K \circ F_K)(M).$$

We sometimes write $\text{HMAC}_d[P]$, HMAC_d , or $\text{HMAC}[P]$ instead of HMAC when we want to make the reliance on the block size and/or an underlying ideal primitive explicit.

⁷ RFC 2104 defines HMAC over strings of bytes, but we chose to use bits to provide more general positive results — all our negative results lift to a setting in which only byte strings are used. Note also that for simplicity we assumed H with domain $\{0, 1\}^*$. In practice hash functions often do have some maximal length (e.g., 2^{64}), and in this case HMAC must be restricted to smaller lengths.

In the following sections, we will therefore analyze the security of HMAC in the sense of being indifferentiable from a keyed RO. As we will see, the story is more involved than one might expect.

4.1 Weak Key Pairs in HMAC

Towards understanding the indifferentiability of HMAC, we start by observing that the way HMAC handles keys gives rise to two worrisome classes of weak key pairs.

COLLIDING KEYS. We say that keys $K \neq K'$ *collide* if $\rho(K) \parallel 0^{d-|\rho(K)|} = \rho(K') \parallel 0^{d-|\rho(K')|}$. For any message M and colliding keys K, K' it holds that $\text{HMAC}(K, M) = \text{HMAC}(K', M)$. Colliding keys exist because of HMAC's ambiguous encoding of different-length keys. Examples of colliding keys include any K, K' for which $|K| < d$ and $K' = K \parallel 0^s$ where $1 \leq s \leq d - |K|$. Or any K, K' such that $|K| > d$ and $K' = H(K)$. As long as H is collision-resistant, two keys of the same length can never collide.

Colliding keys enable a simple attack against indifferentiability. Consider $\text{HMAC}[P]$ for any underlying function P . Then let \mathcal{A} pick two keys $K \neq K'$ that collide and an arbitrary message M . It queries its **Func** oracle on (K, M) and (K', M) to retrieve two values Y, Y' . If $Y = Y'$ then it returns 1 (guessing that it is in game $\text{Real}_{\text{HMAC}[P], \mathcal{R}}$) and returns 0 otherwise (guessing that it is in game $\text{Ideal}_{\mathcal{R}, \mathcal{S}}$). The advantage of \mathcal{A} is equal to $1 - 2^n$ regardless of the simulator \mathcal{S} , which is never invoked.

Note that this result extends directly to rule out related-key attack security [6] of HMAC as a PRF should a related-key function be available that enables deriving colliding keys.

AMBIGUOUS KEYS. A pair of keys $K \neq K'$ is *ambiguous* if $\rho(K) \oplus \text{ipad} = \rho(K') \oplus \text{opad}$. For any X , both $F_K(X) = G_{K'}(X)$ and $G_K(X) = F_{K'}(X)$ when K, K' are ambiguous. An example such pair is K, K' of length d bits for which $K \oplus K' = \text{xpad}$.

For any key K , there exists one key K' that is easily computable and for which K, K' are ambiguous: set $K' = \rho(K) \oplus \text{xpad}$. Finding a third key K'' that is also ambiguous with K is intractable should H be collision resistant. The easily-computable K' will not necessarily have the same length as K . In fact, there exist ambiguous key pairs of the same length k only when $k \in \{d-1, d\}$. For a fixed length shorter than $d-1$, no ambiguous key pairs exist due to the fact that the second least significant bit of xpad is 1. For a fixed length longer than d bits, if $n < d-1$ then no ambiguous key pairs exist and if $n \geq d-1$ then producing ambiguous key pairs would require finding K, K' such that $H(K) \oplus H(K')$ equals the first n bits of xpad . This is intractable for any reasonable hash function H .

Ambiguous key pairs give rise to a chain-shift like property. Let M be some message and K, K' be an ambiguous key pair. Then, we have that $\rho(K') = \rho(K) \oplus \text{xpad}$ and so $F_K(M) = G_{K'}(M)$. Thus,

$$\text{HMAC}(K', F_K(M)) = G_{K'}(\text{HMAC}(K, M)) .$$

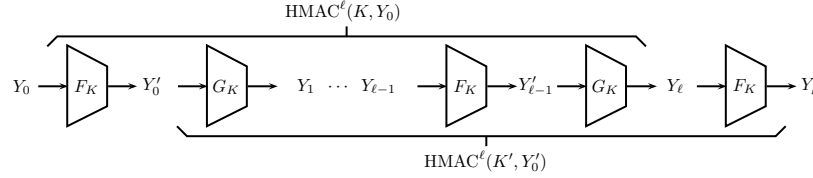


Fig. 3. Diagram of two hash chains $(K, Y) = (Y_0, \dots, Y_\ell)$ and $(K', Y') = (Y'_0, \dots, Y'_\ell)$ for HMAC where $\rho(K') = \rho(K) \oplus \text{xpad}$.

As with H^2 , this property gives rise to problems in the context of hash chains. A hash chain $Y = (K, Y_0, \dots, Y_\ell)$ is a key K , a message Y_0 , and a sequence of ℓ values $Y_i = H(K, Y_{i-1})$ for $1 \leq i \leq \ell$. So a keyed hash chain $Y = (K, Y_0, \dots, Y_\ell)$ for HMAC has $Y_i = \text{HMAC}(K, Y_{i-1})$ for $1 \leq i \leq \ell$. Given K, Y_0, Y_ℓ for a chain $Y = (K, Y_0, \dots, Y_\ell)$, it is easy for an adversary to compute the start and end of a new chain $Y' = (K', Y'_0, \dots, Y'_\ell)$ that does not overlap with Y . See Figure 3. In the full version, we detail how this structure can be abused in the context of an HMAC-based mutual proofs of work protocol. We also give an analogue of Theorem 1, i.e., a lower bound on the indistinguishability of HMAC from a RO when ambiguous key pairs can be queried.

4.2 Indistinguishability of HMAC with Restricted Keys

We have seen that HMAC's construction gives rise to two kinds of weak key pairs that can be abused to show that HMAC is not indistinguishable from a keyed RO (with good bounds). But weak key pairs are serendipitously avoided in most applications. For example, the recommended usage of HKDF [28] specifies keys of a fixed length less than $d - 1$. Neither kind of weak key pairs exist within this subset of the key space.

While one can show indistinguishability for a variety of settings in which weak key pairs are avoided, we focus for simplicity on the case mentioned above. That is, we restrict to keys K for which $|K| = k$ and k is a fixed integer different less than $d - 1$. The full version provides a more general set of results, covering also, for example, use of HMAC with a fixed key of any length less than or equal to d .

As our first positive result, we have the following theorem, which establishes the security of HMAC when modeling the underlying hash function as a RO.

Theorem 3. Fix $d, k, n > 0$ with $k < d - 1$. Let $P: \{0, 1\}^* \rightarrow \{0, 1\}^n$ be a RO, and consider $\text{HMAC}_d[P]$ restricted to k -bit keys. Let $\mathcal{R}: \{0, 1\}^* \times \{0, 1\}^* \rightarrow \{0, 1\}^n$ be a keyed RO. Then there exists a simulator \mathcal{S} such that for any distinguisher \mathcal{A} whose total query cost is σ it holds that

$$\text{Adv}_{\text{HMAC}_d[P], \mathcal{R}, \mathcal{S}}^{\text{indiff}}(\mathcal{A}) \leq \mathcal{O}\left(\frac{\sigma^2}{2^n}\right)$$

\mathcal{S} makes at most q_2 queries and runs in time $\mathcal{O}(q_2 \log q_2)$ where q_2 is the number of Prim queries made by \mathcal{A} . \square

The use of $\mathcal{O}(\cdot)$ just hides small constants. The proof is given in the full version. Combining Theorem 3 with the indifferntiability composition theorem allows us to conclude security for $\text{HMAC}_d[H]$ for underlying hash function H that is, itself, indifferntiable from a RO. For example, should H be one of the proven-indifferntiable SHA-3 candidates. This does not, however, give us a security guarantee should H not be indifferntiable from a RO, as is the case with MD based hash functions. We therefore also prove, in the full version, the following theorem that establishes indifferntiability of HMAC using an underlying hash function built via the strengthened Merkle-Damgård (SMD) domain extension transform.

Theorem 4. *Fix $d, k, n > 0$ with $k < d - 1$ and $d \geq n$. Let $f: \{0, 1\}^n \times \{0, 1\}^d \rightarrow \{0, 1\}^n$ be a RO and consider $\text{HMAC}_d[\text{SMD}[f]]$ restricted to k -bit keys. Let $\mathcal{R}: \{0, 1\}^* \times \{0, 1\}^* \rightarrow \{0, 1\}^n$ be a keyed RO. Then there exists a simulator \mathcal{S} such that for any distinguisher \mathcal{A} whose total query cost is $\sigma \leq 2^{n-2}$ it holds that*

$$\text{Adv}_{\text{HMAC}_d[\text{SMD}[f]], \mathcal{R}, \mathcal{S}}^{\text{indiff}}(\mathcal{A}) \leq \mathcal{O}\left(\frac{\sigma^2}{2^n}\right)$$

\mathcal{S} makes at most q_2 queries and runs in time $\mathcal{O}(q_2 \log q_2)$ where q_2 is the number of Prim queries by \mathcal{A} . \square

We note that the restriction to $\sigma \leq 2^{n-2}$ in the theorem statement is just a technicality to make the bound simpler and likewise the use of $\mathcal{O}(\cdot)$ in the advantage statement hides just a small constant.

Unlike our positive results about H^2 , the bounds provided by Theorems 3 and 4 match, up to small constants, results for other now-standard indifferntiable constructions (c.f., [13]). First, the advantage bounds both hold up to the birthday bound, namely $\sigma \approx 2^{n/2}$. Second, the simulators are efficient and, specifically, make at most one query per invocation. All this enables use of the indifferntiability composition theorem in a way that yields strong, standard concrete security bounds.

Acknowledgments

The authors thank Hugo Krawczyk for providing significant feedback and suggestions, in particular encouraging the authors to include positive results for the indifferntiability of HMAC; Niels Ferguson for in-depth discussions regarding the security of H^2 ; and the anonymous reviewers for their helpful suggestions. Dodis was supported in part by NSF grants CNS-1065134, CNS-1065288, CNS-1017471, CNS-0831299. Ristenpart was supported in part by NSF grant CNS-1065134. Steinberger is supported by the National Basic Research Program of China Grant 2011CBA00300, 2011CBA00301, the National Natural Science Foundation of China Grant 61033001, 61061130540, 61073174, and by NSF grant 0994380. Tessaro was supported in part by NSF grants CCF-0915675, CCF-1018064.

This material is based on research sponsored by DARPA under agreement numbers FA8750-11-C-0096 and FA8750-11-2-0225. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the U.S. Government.

References

1. Elena Andreeva, Bart Mennink, and Bart Preneel. On the indistinguishability of the Grøstl hash function. In Juan A. Garay and Roberto De Prisco, editors, *SCN 10: 7th International Conference on Security in Communication Networks*, volume 6280 of *Lecture Notes in Computer Science*, pages 88–105. Springer, September 2010.
2. Elena Andreeva, Gregory Neven, Bart Preneel, and Thomas Shrimpton. Seven-property-preserving iterated hashing: ROX. In Kaoru Kurosawa, editor, *Advances in Cryptology – ASIACRYPT 2007*, volume 4833 of *Lecture Notes in Computer Science*, pages 130–146. Springer, December 2007.
3. Mihir Bellare. New proofs for NMAC and HMAC: Security without collision-resistance. In Cynthia Dwork, editor, *Advances in Cryptology – CRYPTO 2006*, volume 4117 of *Lecture Notes in Computer Science*, pages 602–619. Springer, August 2006.
4. Mihir Bellare, Zvika Brakerski, Moni Naor, Thomas Ristenpart, Gil Segev, Hovav Shacham, and Scott Yilek. Hedged public-key encryption: How to protect against bad randomness. In Mitsuru Matsui, editor, *Advances in Cryptology – ASIACRYPT 2009*, volume 5912 of *Lecture Notes in Computer Science*, pages 232–249. Springer, December 2009.
5. Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In Neal Koblitz, editor, *Advances in Cryptology – CRYPTO’96*, volume 1109 of *Lecture Notes in Computer Science*, pages 1–15. Springer, August 1996.
6. Mihir Bellare and Tadayoshi Kohno. A theoretical treatment of related-key attacks: RKA-PRPs, RKA-PRFs, and applications. In Eli Biham, editor, *Advances in Cryptology – EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 491–506. Springer, May 2003.
7. Mihir Bellare and Thomas Ristenpart. Multi-property-preserving hash domain extension and the EMD transform. In Xuejia Lai and Kefei Chen, editors, *Advances in Cryptology – ASIACRYPT 2006*, volume 4284 of *Lecture Notes in Computer Science*, pages 299–314. Springer, December 2006.
8. Mihir Bellare and Thomas Ristenpart. Hash functions in the dedicated-key setting: Design choices and MPP transforms. In Lars Arge, Christian Cachin, Tomasz Jurdzinski, and Andrzej Tarlecki, editors, *ICALP 2007: 34th International Colloquium on Automata, Languages and Programming*, volume 4596 of *Lecture Notes in Computer Science*, pages 399–410. Springer, July 2007.
9. Mihir Bellare, Thomas Ristenpart, and Stefano Tessaro. Multi-instance security and its application to password-based cryptography. In *Advances in Cryptology – CRYPTO ’12*, Lecture Notes in Computer Science. Springer, 2012.

10. Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In Serge Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 409–426. Springer, May / June 2006.
11. Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. On the indistinguishability of the sponge construction. In Nigel P. Smart, editor, *Advances in Cryptology – EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 181–197. Springer, April 2008.
12. Donghoon Chang and Mridul Nandi. Improved indistinguishability security analysis of chopMD hash function. In Kaisa Nyberg, editor, *Fast Software Encryption – FSE 2008*, volume 5086 of *Lecture Notes in Computer Science*, pages 429–443. Springer, February 2008.
13. Jean-Sébastien Coron, Yevgeniy Dodis, Cécile Malinaud, and Prashant Puniya. Merkle-Damgård revisited: How to construct a hash function. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 430–448. Springer, August 2005.
14. Yevgeniy Dodis, Rosario Gennaro, Johan Håstad, Hugo Krawczyk, and Tal Rabin. Randomness extraction and key derivation using the CBC, cascade and HMAC modes. In Matthew Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 494–510. Springer, August 2004.
15. Yevgeniy Dodis, Leonid Reyzin, Ronald L. Rivest, and Emily Shen. Indistinguishability of permutation-based compression functions and tree-based modes of operation, with applications to MD6. In Orr Dunkelman, editor, *Fast Software Encryption – FSE 2009*, volume 5665 of *Lecture Notes in Computer Science*, pages 104–121. Springer, February 2009.
16. Yevgeniy Dodis, Thomas Ristenpart, and Thomas Shrimpton. Salvaging Merkle-Damgård for practical applications. In Antoine Joux, editor, *Advances in Cryptology – EUROCRYPT 2009*, volume 5479 of *Lecture Notes in Computer Science*, pages 371–388. Springer, April 2009.
17. Yevgeniy Dodis, Thomas Ristenpart, John Steinberger, and Stefano Tessaro. To Hash or Not to Hash, Again? On the Indistinguishability of the Second Iterate and HMAC, 2012. Full version of this paper. Available from authors’ websites.
18. Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In Ernest F. Brickell, editor, *Advances in Cryptology – CRYPTO’92*, volume 740 of *Lecture Notes in Computer Science*, pages 139–147. Springer, August 1993.
19. Cynthia Dwork, Moni Naor, and Hoeteck Wee. Pebbling and proofs of work. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 37–54. Springer, August 2005.
20. Niels Ferguson and Bruce Schneier. *Practical cryptography*. Wiley, 2003.
21. Pierre-Alain Fouque, David Pointcheval, and Sébastien Zimmer. HMAC is a randomness extractor and applications to TLS. In Masayuki Abe and Virgil Gligor, editors, *ASIACCS 08: 3rd Conference on Computer and Communications Security*, pages 21–32. ACM Press, March 2008.
22. J. Franks, P. Hallam-Baker, J. Hostetler, P. Leach, A. Luotonen, E. Sink, and L. Stewart. An Extension to HTTP: Digest Access Authentication. RFC 2069 (Proposed Standard), January 1997. Obsoleted by RFC 2617.
23. Shoichi Hirose, Je Hong Park, and Aaram Yun. A simple variant of the Merkle-Damgård scheme with a permutation. In Kaoru Kurosawa, editor, *Advances in Cryptology – ASIACRYPT 2007*, volume 4833 of *Lecture Notes in Computer Science*, pages 113–129. Springer, December 2007.

24. Ari Juels and John G. Brainard. Client puzzles: A cryptographic countermeasure against connection depletion attacks. In *ISOC Network and Distributed System Security Symposium – NDSS’99*. The Internet Society, February 1999.
25. Ghassan Karame and Srdjan Capkun. Low-cost client puzzles based on modular exponentiation. In Dimitris Gritzalis, Bart Preneel, and Marianthi Theoharidou, editors, *ESORICS 2010: 15th European Symposium on Research in Computer Security*, volume 6345 of *Lecture Notes in Computer Science*, pages 679–697. Springer, 2010.
26. H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-Hashing for Message Authentication. RFC 2104, February 1997.
27. H. Krawczyk and P. Eronen. Hmac-based extract-and-expand key derivation function (hkdf). RFC 5869 (Proposed Standard), January 2010.
28. Hugo Krawczyk. Cryptographic extraction and key derivation: The HKDF scheme. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 631–648. Springer, August 2010.
29. Ueli M. Maurer, Renato Renner, and Clemens Holenstein. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In Moni Naor, editor, *TCC 2004: 1st Theory of Cryptography Conference*, volume 2951 of *Lecture Notes in Computer Science*, pages 21–39. Springer, February 2004.
30. PKCS #5: Password-based cryptography standard (rfc 2898). RSA Data Security, Inc., September 2000. Version 2.0.
31. Thomas Ristenpart, Hovav Shacham, and Thomas Shrimpton. Careful with composition: Limitations of the indifferentiability framework. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 487–506. Springer, May 2011.
32. Thomas Ristenpart and Scott Yilek. When good randomness goes bad: Virtual machine reset vulnerabilities and hedging deployed cryptography. In *Network and Distributed Systems Security – NDSS ’10*. ISOC, 2010.
33. Douglas Stebila, Lakshmi Kuppusamy, Jothi Rangasamy, Colin Boyd, and Juan Manuel González Nieto. Stronger difficulty notions for client puzzles and denial-of-service-resistant protocols. In Aggelos Kiayias, editor, *Topics in Cryptology – CT-RSA 2011*, volume 6558 of *Lecture Notes in Computer Science*, pages 284–301. Springer, February 2011.
34. Gene Tsudik. Message authentication with one-way hash functions. In *Proceedings IEEE INFOCOM’92*, volume 3, pages 2055–2059. IEEE, 1992.
35. XiaoFeng Wang and Michael K. Reiter. Defending against denial-of-service attacks with puzzle auction. In *IEEE Symposium on Security and Privacy*, pages 78–92, 2003.

Design and Implementation of a Homomorphic-Encryption Library

Shai Halevi Victor Shoup

November 30, 2012

Abstract

We describe the design and implementation of a software library that implements the Brakerski-Gentry-Vaikuntanathan (BGV) homomorphic encryption scheme, along with many optimizations to make homomorphic evaluation runs faster, focusing mostly on effective use of the Smart-Vercauteren ciphertext packing techniques. Our library is written in C++ and uses the NTL mathematical library.

Partially supported by DARPA under agreement number FA8750-11-C-0096. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the U.S. Government.

Also partially supported by the Intelligence Advanced Research Projects Activity (IARPA) via Department of Interior National Business Center (DoI/NBC) contract number D11PC20202. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, DoI/NBC, or the U.S. Government.

Contents

1	The BGV Homomorphic Encryption Scheme	1
1.1	Plaintext Slots	2
1.2	Our Modulus Chain and Double-CRT Representation	2
1.3	Modules in our Library	3
2	The Math Layers	3
2.1	The <code>timing</code> module	3
2.2	<code>NumbTh</code> : Miscellaneous Utilities	3
2.3	<code>bluestein</code> and <code>Cmodulus</code> : Polynomials in FFT Representation	4
2.4	<code>PAgebra</code> : The Structure of \mathbb{Z}_m^* and $\mathbb{Z}_m^*/\langle 2 \rangle$	5
2.5	<code>PAgebraModTwo</code> / <code>PAgebraMod2r</code> : Plaintext Slots	6
2.6	<code>IndexSet</code> and <code>IndexMap</code> : Sets and Indexes	8
2.6.1	The <code>IndexSet</code> class	8
2.6.2	The <code>IndexMap</code> class	9
2.7	<code>FHEcontext</code> : Keeping the parameters	9
2.8	<code>DoubleCRT</code> : Efficient Polynomial Arithmetic	10
3	The Crypto Layer	13
3.1	The <code>Ctxt</code> module: Ciphertexts and homomorphic operations	13
3.1.1	The <code>SKHandle</code> class	14
3.1.2	The <code>CtxtPart</code> class	15
3.1.3	The <code>Ctxt</code> class	15
3.1.4	Noise estimate	16
3.1.5	Modulus-switching operations	18
3.1.6	Key-switching/re-linearization	19
3.1.7	Native arithmetic operations	21
3.1.8	More <code>Ctxt</code> methods	23
3.2	The <code>FHE</code> module: Keys and key-switching matrices	24
3.2.1	The <code>KeySwitch</code> class	24
3.2.2	The <code>FHEPubKey</code> class	25
3.2.3	The <code>FHESecKey</code> class	27
3.3	The <code>KeySwitching</code> module: What matrices to generate	28
4	The Data-Movement Layer	29
4.1	The classes <code>EncryptedArray</code> and <code>EncryptedArrayMod2r</code>	29
5	Using the Library	33
5.1	Homomorphic Operations over $GF(2^8)$	34
5.2	Homomorphic Operations over \mathbb{Z}_{25}	35
A	Proof of noise-estimate	37

Organization of This Report

We begin in Section 1 with a brief high-level overview of the BGV cryptosystem and some important features of the variant that we implemented and our choice of representation, as well as an overview of the structure of our library. Then in Sections 2, 3, 4 we give a bottom-up detailed description of all the modules in the library. We conclude in Section 5 with some examples of using this library.

1 The BGV Homomorphic Encryption Scheme

A homomorphic encryption scheme [8, 3] allows processing of encrypted data even without knowing the secret decryption key. In this report we describe the design and implementation of a software library that we wrote to implement the Brakerski-Gentry-Vaikuntanathan (BGV) homomorphic encryption scheme [2]. We begin by a high-level description of the the BGV variant that we implemented, followed by a detailed description of the various software components in our implementation. the description in this section is mostly taken from the full version of [5].

Below we denote by $[\cdot]_q$ the reduction-mod- q function, namely mapping an integer $z \in \mathbb{Z}$ to the unique representative of its equivalence class modulo q in the interval $(-q/2, q/2]$. We use the same notation for modular reduction of vectors, matrices, and polynomials (in coefficient representation).

Our BGV variant is defined over polynomial rings of the form $\mathbb{A} = \mathbb{Z}[X]/\Phi_m(X)$ where m is a parameter and $\Phi_m(X)$ is the m 'th cyclotomic polynomial. The “native” plaintext space for this scheme is usually the ring $\mathbb{A}_2 = \mathbb{A}/2\mathbb{A}$, namely binary polynomials modulo $\Phi_m(X)$. (Our implementation supports other plaintext spaces as well, but in this report we mainly describe the case of plaintext space \mathbb{A}_2 . See some more details in Section 2.4.) We use the Smart-Vercauteren CTR-based encoding technique [10] to “pack” a vector of bits in a binary polynomial, so that polynomial arithmetic in \mathbb{A}_2 translates to entry-wise arithmetic on the packed bits.

The ciphertext space for this scheme consists of vectors over $\mathbb{A}_q = \mathbb{A}/q\mathbb{A}$, where q is an odd modulus that evolves with the homomorphic evaluation. Specifically, the system is parametrized by a “chain” of moduli of decreasing size, $q_0 > q_1 > \dots > q_L$ and freshly encrypted ciphertexts are defined over R_{q_0} . During homomorphic evaluation we keep switching to smaller and smaller moduli until we get ciphertexts over \mathbb{A}_{q_L} , on which we cannot compute anymore. We call ciphertexts that are defined over \mathbb{A}_{q_i} “level- i ciphertexts”. These level- i ciphertexts are 2-element vectors over R_{q_i} , i.e., $\vec{c} = (c_0, c_1) \in (\mathbb{A}_{q_i})^2$.

Secret keys are polynomials $\mathfrak{s} \in \mathbb{A}$ with “small” coefficients, and we view \mathfrak{s} as the second element of the 2-vector $\vec{s} = (1, \mathfrak{s})$. A level- i ciphertext $\vec{c} = (c_0, c_1)$ encrypts a plaintext polynomial $m \in \mathbb{A}_2$ with respect to $\vec{s} = (1, \mathfrak{s})$ if we have the equality over \mathbb{A} , $[\langle \vec{c}, \vec{s} \rangle]_{q_i} = [c_0 + \mathfrak{s} \cdot c_1]_{q_i} \equiv m \pmod{2}$, and moreover the polynomial $[c_0 + \mathfrak{s} \cdot c_1]_{q_i}$ is “small”, i.e. all its coefficients are considerably smaller than q_i . Roughly, that polynomial is considered the “noise” in the ciphertext, and its coefficients grow as homomorphic operations are performed. We note that the crux of the noise-control technique from [2] is that a level- i ciphertext can be publicly converted into a level- $(i+1)$ ciphertext (with respect to the same secret key), and that this transformation reduces the noise in the ciphertext roughly by a factor of q_{i+1}/q_i .

Following [7, 4, 5], we think of the “size” of a polynomial $a \in \mathbb{A}$ the norm of its canonical embedding. Recall that the canonical embedding of $a \in \mathbb{A}$ into $\mathbb{C}^{\phi(m)}$ is the $\phi(m)$ -vector of complex numbers $\sigma(a) = (a(\tau_m^j))_j$ where τ_m is a complex primitive m -th root of unity ($\tau_m = e^{2\pi i/m}$) and the indexes j range over all of \mathbb{Z}_m^* . We denote the l_2 -norm of the canonical embedding of a by

$\|a\|_2^{\text{canon}}$.

The basic operations that we have in this scheme are the usual key-generation, encryption, and decryption, the homomorphic evaluation routines for addition, multiplication and automorphism (and also addition-of-constant and multiplication-by-constant), and the “ciphertext maintenance” operations of key-switching and modulus-switching. These are described in the rest of this report, but first we describe our plaintext encoding conventions and our Double-CRT representation of polynomials.

1.1 Plaintext Slots

The native plaintext space of our variant of BGV are elements of \mathbb{A}_2 , and the polynomial $\Phi_m(X)$ factors modulo 2 into ℓ irreducible factors, $\Phi_m(X) = F_1(X) \cdot F_2(X) \cdots F_\ell(X) \pmod{2}$, all of degree $d = \phi(m)/\ell$. Just as in [2, 4, 10] each factor corresponds to a “plaintext slot”. That is, we can view a polynomial $a \in \mathbb{A}_2$ as representing an ℓ -vector $(a \bmod F_i)_{i=1}^\ell$.

More specifically, for the purpose of packing we think of a polynomial $a \in \mathbb{A}_2$ not as a binary polynomial but as a polynomial over the extension field \mathbb{F}_{2^d} (with some specific representation), and the plaintext values that are encoded in a are its evaluations at ℓ specific primitive m -th roots of unity in \mathbb{F}_{2^d} . In other words, if $\rho \in \mathbb{F}_{2^d}$ is a particular fixed primitive m -th root of unity, and our distinguished evaluation points are $\rho^{t_1}, \rho^{t_2}, \dots, \rho^{t_\ell}$ (for some set of indexes $T = \{t_1, \dots, t_\ell\}$), then the vector of plaintext values encoded in a is:

$$(a(\rho^{t_j}) : t_j \in T).$$

See Section 2.4 for a discussion of the choice of representation of \mathbb{F}_{2^d} and the evaluation points.

It is standard fact that the Galois group $\mathcal{G} = \mathcal{G}(\mathbb{Q}(\rho_m)/\mathbb{Q})$ consists of the mappings $\kappa_k : a(X) \mapsto a(X^k) \bmod \Phi_m(X)$ for all k co-prime with m , and that it is isomorphic to \mathbb{Z}_m^* . As noted in [4], for each $i, j \in \{1, 2, \dots, \ell\}$ there is an element $\kappa_k \in \mathcal{G}$ which sends an element in slot i to an element in slot j . Indeed if we set $k = t_j^{-1} \cdot t_i \pmod{m}$ and $b = \kappa_k(a)$ then we have

$$b(\rho^{t_j}) = a(\rho^{t_j^k}) = a(\rho^{t_j \cdot t_j^{-1} t_i}) = a(\rho^{t_i}),$$

so the element in the j ’th slot of b is the same as that in the i ’th slot of a . In addition to these “data-movement maps”, \mathcal{G} contains also the Frobenius maps, $X \rightarrow X^{2^i}$, which also act as Frobenius on the individual slots separately.

We note that the values that are encoded in the slots do not have to be individual bits, in general they can be elements of the extension field \mathbb{F}_{2^d} (or any sub-field of it). For example, for the AES application we may want to pack elements of \mathbb{F}_{2^8} in the slots, so we choose the parameters so that \mathbb{F}_{2^8} is a sub-field of \mathbb{F}_{2^d} (which means that d is divisible by 8).

1.2 Our Modulus Chain and Double-CRT Representation

We define the chain of moduli by choosing $L + 1$ “small primes” p_0, p_1, \dots, p_L and the l ’th modulus in our chain is defined as $q_l = \prod_{j=0}^l p_j$. The primes p_i ’s are chosen so that for all i , $\mathbb{Z}/p_i\mathbb{Z}$ contains a primitive m -th root of unity (call it ζ_i) so $\Phi_m(X)$ factors modulo p_i to linear terms $\Phi_m(X) = \prod_{j \in \mathbb{Z}_m^*} (X - \zeta_i^j) \pmod{p_i}$.

A key feature of our implementation is that we represent an element $a \in \mathbb{A}_{q_l}$ via double-CRT representation, with respect to both the integer factors of q_l and the polynomial factor of $\Phi_m(X)$

mod q_l . A polynomial $a \in \mathbb{A}_q$ is represented as the $(l+1) \times \phi(m)$ matrix of its evaluation at the roots of $\Phi_m(X)$ modulo p_i for $i = 0, \dots, l$:

$$\text{DoubleCRT}^l(a) = \left(a(\zeta_i^j) \bmod p_i \right)_{0 \leq i \leq l, j \in \mathbb{Z}_m^*}.$$

Addition and multiplication in \mathbb{A}_q can be computed as component-wise addition and multiplication of the entries in the two tables (modulo the appropriate primes p_i),

$$\begin{aligned} \text{DoubleCRT}^l(a+b) &= \text{DoubleCRT}^l(a) + \text{DoubleCRT}^l(b), \\ \text{DoubleCRT}^l(a \cdot b) &= \text{DoubleCRT}^l(a) \cdot \text{DoubleCRT}^l(b). \end{aligned}$$

Also, for an element of the Galois group $\kappa \in \mathcal{G}\text{al}$, mapping $a(X) \in \mathbb{A}$ to $a(X^k) \bmod \Phi_m(X)$, we can evaluate $\kappa(a)$ on the double-CRT representation of a just by permuting the columns in the matrix, sending each column j to column $j \cdot k \bmod m$.

1.3 Modules in our Library

Very roughly, our HE library consists of four layers: in the bottom layer we have modules for implementing mathematical structures and various other utilities, the second layer implements our Double-CRT representation of polynomials, the third layer implements the cryptosystem itself (with the “native” plaintext space of binary polynomials), and the top layer provides interfaces for using the cryptosystem to operate on arrays of plaintext values (using the plaintext slots as described in Section 1.1). We think of the bottom two layers as the “math layers”, and the top two layers as the “crypto layers”, and describe them in detail in Sections 2 and 3, respectively. A block-diagram description of the library is given in Figure 1. Roughly, the modules `NumbTh`, `timing`, `bluestein`, `PAAlgebra`, `PAAlgebraModTwo`, `PAAlgebraMod2r`, `Cmodulus`, `IndexSet` and `IndexMap` belong to the bottom layer, `FHEcontext`, `SingleCRT` and `DoubleCRT` belong to the second layer, `FHE`, `Ctxt` and `KeySwitching` are in the third layer, and `EncryptedArray` and `EncryptedArrayMod2r` are in the top layer.

2 The Math Layers

2.1 The timing module

This module contains some utility function for measuring the time that various methods take to execute. To use it, we insert the macro `FHE_TIMER_START` at the beginning of the method(s) that we want to time and `FHE_TIMER_STOP` at the end, then the main program needs to call the function `setTimersOn()` to activate the timers and `setTimersOff()` to pause them. We can have at most one timer per method/function, and the timer is called by the same name as the function itself (using the pre-defined variable `_func_`). To obtain the value of a given timer (in seconds), the application can use the function `double getTime4func(const char *fncName)`, and the function `printAllTimers()` prints the values of all timers to the standard output.

2.2 NumbTh: Miscellaneous Utilities

This module started out as an implementation of some number-theoretic algorithms (hence the name), but since then it grew to include many different little utility functions. For example, CRT-

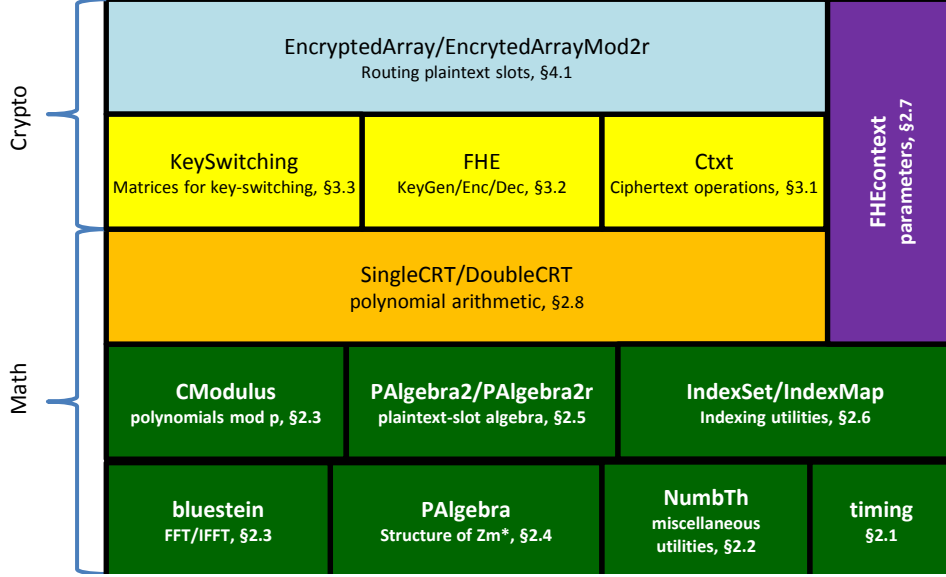


Figure 1: A block diagram of the Homomorphic-Encryption library

reconstruction of polynomials in coefficient representation, conversion functions between different types, procedures to sample at random from various distributions, etc.

2.3 bluestein and Cmodulus: Polynomials in FFT Representation

The `bluestein` module implements a non-power-of-two FFT over a prime field \mathbb{Z}_p , using the Bluestein FFT algorithm [1]. We use modulo- p polynomials to encode the FFTs inputs and outputs. Specifically this module builds on Shoup’s NTL library [9], and contains both a bigint version with types `ZZ_p` and `ZZ_pX`, and a smallint version with types `zz_p` and `zz_pX`. We have the following functions:

```
void BluesteinFFT(ZZ_pX& x, const ZZ_pX& a, long n, const ZZ_p& root,
                 ZZ_pX& powers, FFTRep& Rb);
void BluesteinFFT(zz_pX& x, const zz_pX& a, long n, const zz_p& root,
                 zz_pX& powers, fftRep& Rb);
```

These functions compute length- n FFT of the coefficient-vector of `a` and put the result in `x`. If the degree of `a` is less than n then it treats the top coefficients as 0, and if the degree is more than n then the extra coefficients are ignored. Similarly, if the top entries in `x` are zeros then `x` will have degree smaller than n . The argument `root` needs to be a $2n$ -th root of unity in \mathbb{Z}_p . The inverse-FFT is obtained just by calling `BluesteinFFT(...,root-1,...)`, but this procedure is *NOT SCALED*. Hence calling `BluesteinFFT(x,a,n,root,...)` and then `BluesteinFFT(b,x,n,root-1,...)` will result in having `b = n × a`.

In addition to the size- n FFT of `a` which is returned in `x`, this procedure also returns the powers of `root` in the `powers` argument, `powers = (1,root,root4,root9,...,root(n-1)2)`. In the `Rb` argument it returns the size- N FFT representation of the negative powers, for some $N \geq 2n - 1$, N a power of two:

$$\mathbf{Rb} = FFT_N(0, \dots, 0, \text{root}^{-(n-1)^2}, \dots, \text{root}^{-4}, \text{root}^{-1}, 1, \text{root}^{-1}, \text{root}^{-4}, \dots, \text{root}^{-(n-1)^2}, 0, \dots, 0).$$

On subsequent calls with the same `powers` and `Rb`, these arrays are not computed again but taken from the pre-computed arguments. If the `powers` and `Rb` arguments are initialized, then it is assumed that they were computed correctly from `root`. The behavior is undefined when calling with initialized `powers` and `Rb` but a different `root`. (In particular, to compute the inverse-FFT using `root-1`, one must provide different `powers` and `Rb` arguments than those that were given when computing in the forward direction using `root`.) This procedure *cannot* be used for in-place FFT, calling `BluesteinFFT(x, x, ...)` will just zero-out the polynomial `x`.

The classes `Cmodulus` and `CModulus`. These classes provide an interface layer for the FFT routines above, relative to a single prime (where `Cmodulus` is used for smallint primes and `CModulus` for bigint primes). They keep the NTL “current modulus” structure for that prime, as well as the `powers` and `Rb` arrays for FFT and inverse-FFT under that prime. They are constructed with the constructors

```
Cmodulus(const PAlgebra& ZmStar, const long& q, const long& root);
CModulus(const PAlgebra& ZmStar, const ZZ& q, const ZZ& root);
```

where `ZmStar` described the structure of \mathbb{Z}_m^* (see Section 2.4), `q` is the prime modulus and `root` is a primitive $2m$ -’th root of unity modulo `q`. (If the constructor is called with `root = 0` then it computes a $2m$ -th root of unity by itself.) Once an object of one of these classes is constructed, it provides an FFT interfaces via

```
void Cmodulus::FFT(vec_long& y, const ZZx& x) const; // y = FFT(x)
void Cmodulus::iFFT(ZZx& x, const vec_long& y) const; // x = FFT-1(y)
```

(And similarly for `CModulus` using `vec_ZZ` instead of `vec_long`). These method are inverses of each other. The methods of these classes affect the NTL “current modulus”, and it is the responsibility of the caller to backup and restore the modulus if needed (using the NTL constructs `zz_pBak/ZZ_pBak`).

2.4 PAlgebra: The Structure of \mathbb{Z}_m^* and $\mathbb{Z}_m^*/\langle 2 \rangle$

The class `PAlgebra` is the base class containing the structure of \mathbb{Z}_m^* , as well as the quotient group $\mathbb{Z}_m^*/\langle 2 \rangle$. We represent \mathbb{Z}_m^* as $\mathbb{Z}_m^* = \langle 2 \rangle \times \langle g_1, g_2, \dots \rangle \times \langle h_1, h_2, \dots \rangle$, where the g_i ’s have the same order in \mathbb{Z}_m^* as in $\mathbb{Z}_m^*/\langle 2 \rangle$, and the h_i ’s generate the group $\mathbb{Z}_m^*/\langle 2, g_1, g_2, \dots \rangle$ and they do not have the same order in \mathbb{Z}_m^* as in $\mathbb{Z}_m^*/\langle 2 \rangle$.

We compute this representation in a manner similar (but not identical) to the proof of the fundamental theorem of finitely generated abelian groups. Namely we keep the elements in equivalence classes of the “quotient group so far”, and each class has a representative element (called a pivot), which in our case we just choose to be the smallest element in the class. Initially each element is in its own class. At every step, we choose the highest order element g in the current quotient group and add it as a new generator, then unify classes if their members are a factor of g from each other, repeating this process until no further unification is possible. Since we are interested in the quotient group $\mathbb{Z}_m^*/\langle 2 \rangle$, we always choose 2 as the first generator.

One twist in this routine is that initially we only choose an element as a new generator if its order in the current quotient group is the same as in the original group \mathbb{Z}_m^* . Only after no such elements are available, do we begin to use generators that do not have the same order as in \mathbb{Z}_m^* .

Once we chose all the generators (and for each generator we compute its order in the quotient group where it was chosen), we compute a set of “slot representatives” as follows: Putting all the

g_i 's and h_i 's in one list, let us denote the generators of $\mathbb{Z}_m^*/\langle 2 \rangle$ by $\{f_1, f_2, \dots, f_n\}$, and let $\text{ord}(f_i)$ be the order of f_i in the quotient group at the time that it was added to the list of generators. The slot-index representative set is

$$T \stackrel{\text{def}}{=} \left\{ \prod_{i=1}^n f_i^{e_i} \bmod m : \forall i, e_i \in \{0, 1, \dots, \text{ord}(f_i) - 1\} \right\}.$$

Clearly, we have $T \subset \mathbb{Z}_m^*$, and moreover T contains exactly one representative from each equivalence class of $\mathbb{Z}_m^*/\langle 2 \rangle$. Recall that we use these representatives in our encoding of plaintext slots, where a polynomial $a \in \mathbb{A}_2$ is viewed as encoding the vector of \mathbb{F}_{2^d} elements $(a(\rho^t) \in \mathbb{F}_{2^d} : t \in T)$, where ρ is some fixed primitive m -th root of unity in \mathbb{F}_{2^d} .

In addition to defining the sets of generators and representatives, the class `PAgebra` also provides translation methods between representations, specifically:

```
int ith_rep(unsigned i) const;
    Returns  $t_i$ , i.e., the  $i$ 'th representative from  $T$ .

int indexOfRep(unsigned t) const;
    Returns the index  $i$  such that  $\text{ith\_rep}(i) = t$ .

int exponentiate(const vector<unsigned>& exps, bool onlySameOrd=false) const;
    Takes a vector of exponents,  $(e_1, \dots, e_n)$  and returns  $t = \prod_{i=1}^n f_i^{e_i} \in T$ .

const int* dLog(unsigned t) const;
    On input some  $t \in T$ , returns the discrete-logarithm of  $t$  with the  $f_i$ 's are bases. Namely, a vector  $\text{exps} = (e_1, \dots, e_n)$  such that  $\text{exponentiate}(\text{exps}) = t$ , and moreover  $0 \leq e_i \leq \text{ord}(f_i)$  for all  $i$ .
```

2.5 PAgebraModTwo/PAgebraMod2r: Plaintext Slots

These two classes implements the structure of the plaintext spaces, either $\mathbb{A}_2 = \mathbb{A}/2\mathbb{A}$ (when using mod-2 arithmetic for the plaintext space) or $\mathbb{A}_{2^r} = \mathbb{A}/2^r\mathbb{A}$ (when using mod- 2^r arithmetic, for some small value of r , e.g. mod-128 arithmetic). We typically use the mod-2 arithmetic for real computation, but we expect to use the mod- 2^r arithmetic for bootstrapping, as described in [6]. Below we cover the mod-2 case first, then extend it to mod- 2^r .

For the mod-2 case, the plaintext slots are determined by the factorization of $\Phi_m(X)$ modulo 2 into ℓ degree- d polynomials. Once we have that factorization, $\Phi_m(X) = \prod_j F_j(X) \pmod{2}$, we choose an arbitrary factor as the “first factor”, denote it $F_1(X)$, and this corresponds to the first input slot (whose representative is $1 \in T$). With each representative $t \in T$ we then associate the factor $\text{GCD}(F_1(X^t), \Phi_m(X))$, with polynomial-GCD computed modulo 2. Note that fixing a representation of the field $\mathbb{K} = \mathbb{Z}_2[X]/F_1(X) \cong \mathbb{F}_{2^d}$ and letting ρ be a root of F_1 in \mathbb{K} , we get that the factor associated with the representative t is the minimal polynomial of $\rho^{1/t}$. Yet another way of saying the same thing, if the roots of F_1 in \mathbb{K} are $\rho, \rho^2, \rho^4, \dots, \rho^{2^{d-1}}$ then the roots of the factor associated to t are $\rho^{1/t}, \rho^{2/t}, \rho^{4/t}, \dots, \rho^{2^{d-1}/t}$, where the arithmetic in the exponent is modulo m .

After computing the factors of $\Phi_m(X)$ modulo 2 and the correspondence between these factors and the representatives from T , the class `PAgebraModTwo` provide encoding/decoding methods to pack elements in polynomials and unpack them back. Specifically we have the following methods:

```
void mapToSlots(vector<GF2X>& maps, const GF2X& G) const;
```

Computes the mapping between base- G representation and representation relative to the slot polynomials. (See more discussion below.)

```
void embedInSlots(GF2X&a, const vector<GF2X>&alphas, const vector<GF2X>&maps) const;
```

Use the maps that were computed in `mapToSlots` to embeds the plaintext values in `alphas` into the slots of the polynomial $a \in \mathbb{A}_2$. Namely, for every plaintext slot i with representative $t_i \in T$, we have $a(\rho^{t_i}) = \text{alphas}[i]$. Note that `alphas[i]` is an element in base- G representation, while $a(\rho^t)$ is computed relative to the representation of \mathbb{F}_{2^d} as $\mathbb{Z}_2[X]/F_1(X)$. (See more discussion below.)

```
void decodePlaintext(vector<GF2X>& alphas, const GF2X& a,
                    const GF2X& G, const vector<GF2X>& maps) const;
```

This is the inverse of `embedInSlots`, it returns in `alphas` a vector of base- G elements such that `alphas[i] = a(ρti)`.

```
void CRT.decompose(vector<GF2X>& crt, const GF2X& p) const;
```

Returns a vector of polynomials such that `crt[i] = p mod Fti` (with t_i being the i 'th representative in T).

```
void CRT.reconstruct(GF2X& p, vector<GF2X>& crt) const;
```

Returns a polynomial $p \in \mathbb{A}_2$ s.t. for every $i < \ell$ and $t_i = T[i]$, we have $p \equiv \text{crt}[i] \pmod{F_t}$.

The use of the first three functions may need some more explanation. As an illustrative example, consider the case of the AES computation, where we embed bytes of the AES state in the slots, considered as elements of \mathbb{F}_{2^8} relative to the AES polynomial $G(X) = X^8 + X^4 + X^3 + X + 1$. We choose our parameters so that we have $8|d$ (where d is the order of 2 in \mathbb{Z}_m^*), and then use the functions above to embed the bytes into our plaintext slots and extract them back.

We first call `mapToSlots(maps, G)` to prepare compute the mapping from the base- G representation that we use for AES to the “native” representation of four cryptosystem (i.e., relative to F_1 , which is one of the degree- d factors of $\Phi_m(X)$). Once we have `maps`, we use them to embed bytes in a polynomial with `embedInSlots`, and to decode them back with `decodePlaintext`.

The case of plaintext space modulo 2^r , implemented in the class `PAAlgebraMod2r`, is similar. The partition to factors of $\Phi_m(X)$ modulo 2^r and their association with representatives in T is done similarly, by first computing everything modulo 2, then using Hensel lifting to lift into a factorization modulo 2^r . In particular we have the same number ℓ of factors of the same degree d . One difference between the two classes is that when working modulo-2 we can have elements of an extension field \mathbb{F}_{2^d} in the slots, but when working modulo 2^r we enforce the constraint that the slots contain only scalars (i.e., r -bit signed integers, in the range $[-2^{r-1}, 2^{r-1})$). This means that the polynomial G that we use for the representation of the plaintext values is set to the linear polynomial $G(X) = X$. Other than this change, the methods for `PAAlgebraMod2r` are the same as these of `PAAlgebraModTwo`, except that we use the NTL types `zz_p` and `zz_pX` rather than `GF2` and `GF2X`. The methods of the `PAAlgebraMod2r` class affect the NTL “current modulus”, and it is the responsibility of the caller to backup and restore the modulus if needed (using the NTL constructs `zz_pBak/ZZ_pBak`).

2.6 IndexSet and IndexMap: Sets and Indexes

In our implementation, all the polynomials are represented in double-CRT format, relative to some subset of the small primes in our list (cf. Section 1.2). The subset itself keeps changing throughout the computation, and so we can have the same polynomial represented at one point relative to many primes, then a small number of primes, then many primes again, etc. (For example see the implementation of key-switching in Section 3.1.6.) To provide flexibility with these transformations, the `IndexSet` class implements an arbitrary subset of integers, and the `IndexMap` class implements a collection of data items that are indexed by such a subset.

2.6.1 The IndexSet class

The `IndexSet` class implements all the standard interfaces of the abstract data-type of a set, along with just a few extra interfaces that are specialized to sets of *integers*. It uses the standard C++ container `vector<bool>` to keep the actual set, and provides the following methods:

Constructors. The constructors `IndexSet()`, `IndexSet(long j)`, and `IndexSet(long low, long high)`, initialize an empty set, a singleton, and an interval, respectively.

Empty sets and cardinality. The static method `IndexSet::emptySet()` provides a read-only access to an empty set, and the method `s.clear()` removes all the elements in `s`, which is equivalent to `s=IndexSet::emptySet()`.

The method `s.card()` returns the number of elements in `s`.

Traversing a set. The methods `s.first()` and `s.last()` return the smallest and largest element in the set, respectively. For an empty set `s`, `s.first()` returns 0 and `s.last()` returns -1 .

The method `s.next(j)` return the next element after `j`, if any; otherwise `j + 1`. Similarly `s.prev(j)` return the previous element before `j`, if any; otherwise `j - 1`. With these methods, we can iterate through a set `s` using one of:

```
for (long i = s.first(); i <= s.last(); i = s.next(i)) ...
for (long i = s.last(); i >= s.first(); i = s.prev(i)) ...
```

Comparison and membership methods. `operator==` and `operator!=` are provided to test for equality, whereas `s1.disjointFrom(s2)` and its synonym `disjoint(s1,s2)` test if the two sets are disjoint. Also, `s.contains(j)` returns `true` if `s` contains the element `j`, `s.contains(other)` returns `true` if `s` is a superset of `other`. For convenience, the operators `<=`, `<`, `>=` and `>` are also provided for testing the subset relation between sets.

Set operations. The method `s.insert(j)` inserts the integer `j` if it is not in `s`, and `s.remove(j)` removes it if it is there.

Similarly `s1.insert(s2)` returns in `s1` the union of the two sets, and `s1.remove(s2)` returns in `s1` the set difference `s1 \ s2`. Also, `s1.retain(s2)` returns in `s1` the intersection of the two sets. For convenience we also provide the operators `s1|s2` (union), `s1&s2` (intersection), `s1^s2` (symmetric difference, aka xor), and `s1/s2` (set difference).

2.6.2 The IndexMap class

The class template `IndexMap<T>` implements a map of elements of type `T`, indexed by a dynamic `IndexSet`. Additionally, it allows new elements of the map to be initialized in a flexible manner, by providing an initialization function which is called whenever a new element (indexed by a new index j) is added to the map.

Specifically, we have a helper class template `IndexMapInit<T>` that stores a pointer to an initialization function, and possibly also other parameters that the initialization function needs. We then provide a constructor `IndexMap(IndexMapInit<T>* initObject=NULL)` that associates the given initialization object with the new `IndexMap` object. Thereafter, When a new index j is added to the index set, an object `t` of type `T` is created using the default constructor for `T`, after which the function `initObject->init(t)` is called.

In our library, we use an `IndexMap` to store the rows of the matrix of a Double-CRT object. For these objects we have an initialization object that stores the value of $\phi(m)$, and the initialization function, which is called whenever we add a new row, ensures that all the rows have length exactly $\phi(m)$.

After initialization an `IndexMap` object provides the operator `map[i]` to access the type-`T` object indexed by i (if i currently belongs to the `IndexSet`), as well as the methods `map.insert(i)` and `map.remove(i)` to insert or delete a single data item indexed by i , and also `map.insert(s)` and `map.remove(s)` to insert or delete a collection of data items indexed by the `IndexSet` `s`.

2.7 FHEcontext: Keeping the parameters

Objects in higher layers of our library are defined relative to some parameters, such as the integer parameter m (that defines the groups \mathbb{Z}_m^* and $\mathbb{Z}_m^*/\langle 2 \rangle$ and the ring $\mathbb{A} = \mathbb{Z}[X]/\Phi_m(X)$) and the sequence of small primes that determine our modulus-chain. To allow convenient access to these parameters, we define the class `FHEcontext` that keeps them all and provides access methods and some utility functions.

One thing that's included in `FHEcontext` is a vector of `Cmodulus` objects, holding the small primes that define our modulus chain:

```
vector<Cmodulus> moduli; // Cmodulus objects for the different primes
```

We provide access to the `Cmodulus` objects via `context.ithModulus(i)` (that returns a reference of type `const Cmodulus&`), and to the small primes themselves via `context.ithPrime(i)` (that returns a `long`). The `FHEcontext` includes also the various algebraic structures for plaintext arithmetic, specifically we have the three data members:

```
PAAlgebra zMstar;          // The structure of  $\mathbb{Z}_m^*$ 
PAAlgebraModTwo modTwo;    // The structure of  $\mathbb{Z}[X]/(\Phi_m(X), 2)$ 
PAAlgebraMod2r mod2r;      // The structure of  $\mathbb{Z}[X]/(\Phi_m(X), 2^r)$ 
```

In addition to the above, the `FHEcontext` contains a few `IndexSet` objects, describing various partitions of the index-set in the vector of moduli. These partitions are used when generating the key-switching matrices in the public key, and when using them to actually perform key-switching on ciphertexts.

One such partition is “ciphertext” vs. “special” primes: Freshly encrypted ciphertexts are encrypted relative to a subset of the small primes, called the *ciphertext primes*. All other primes are only used during key-switching, these are called the *special primes*. The ciphertext primes, in

turn, are sometimes partitioned further into a number of “digits”, corresponding to the columns in our key-switching matrices. (See the explanation of this partition in Section 3.1.6.) These subsets are stored in the following data members:

```
IndexSet ctxtPrimes;      // the ciphertext primes
IndexSet specialPrimes;  // the "special" primes
vector<IndexSet> digits;  // digits of ctxt/columns of key-switching matrix
```

The `FHEcontext` class provides also some convenience functions for computing the product of a subset of small primes, as well as the “size” of that product (i.e., its logarithm), via the methods:

```
ZZ productOfPrimes(const IndexSet& s) const;
void productOfPrimes(ZZ& p, const IndexSet& s) const;

double logOfPrime(unsigned i) const;    // = log(ithPrime(i))
double logOfProduct(const IndexSet& s) const;
```

Finally, the `FHEcontext` module includes some utility functions for adding moduli to the chain. The method `addPrime(long p, bool isSpecial)` adds a single prime p (either “special” or not), after checking that p has $2m$ ’th roots of unity and it is not already in the list. Then we have three higher-level functions:

```
double AddPrimesBySize(FHEcontext& c, double size, bool special=false);
    Adds to the chain primes whose product is at least exp(size), returns the natural logarithm
    of the product of all added primes.

double AddPrimesByNumber(FHEcontext& c, long n, long atLeast=1, bool special=false);
    Adds  $n$  primes to the chain, all at least as large as the atLeast argument, returns the natural
    logarithm of the product of all added primes.

void buildModChain(FHEcontext& c, long d, long t=3);
    Build modulus chain for a circuit of depth  $d$ , using  $t$  digits in key-switching. This function
    puts  $d$  ciphertext primes in the moduli vector, and then as many “special” primes as needed
    to mod-switch fresh ciphertexts (see Section 3.1.6).
```

2.8 DoubleCRT: Efficient Polynomial Arithmetic

The heart of our library is the `DoubleCRT` class that manipulates polynomials in Double-CRT representation. A `DoubleCRT` object is tied to a specific `FHEcontext`, and at any given time it is defined relative to a subset of small primes from our list, $S \subseteq [0, \dots, \text{context.moduli.size()} - 1]$. Denoting the product of these small primes by $q = \prod_{i \in S} p_i$, a `DoubleCRT` object represents a polynomial $a \in \mathbb{A}_q$ by a matrix with $\phi(m)$ columns and one row for each small prime p_i (with $i \in S$). The i ’th row contains the FFT representation of a modulo p_i , i.e. the evaluations $\{[a(\zeta_i^j)]_{p_i} : j \in \mathbb{Z}_m^*\}$, where ζ_i is some primitive m -th root of unity modulo p_i .

Although the `FHEcontext` must remain fixed throughout, the set S of primes can change dynamically, and so the matrix can lose some rows and add other ones as we go. We thus keep these rows in a dynamic `IndexMap` data member, and the current set of indexes S is available via the method `getIndexSet()`. We provide the following methods for changing the set of primes:

```
void addPrimes(const IndexSet& s);
```

Expand the index set by s . It is assumed that s is disjoint from the current index set. This is an expensive operation, as it needs to convert to coefficient representation and back, in order to determine the values in the added rows.

```
double addPrimesAndScale(const IndexSet& S);
```

Expand the index set by S , and multiply by $q_{\text{diff}} = \prod_{i \in S} p_i$. The set S is assumed to be disjoint from the current index set. Returns $\log(q_{\text{diff}})$. This operation is typically much faster than `addPrimes`, since we can fill the added rows with zeros.

```
void removePrimes(const IndexSet& s);
```

Remove the primes p_i with $i \in s$ from the current index set.

```
void scaleDownToSet(const IndexSet& s, long ptxtSpace);
```

This is a modulus-switching operation. Let Δ be the set of primes that are removed, $\Delta = \text{getIndexSet}() \setminus s$, and $q_{\text{diff}} = \prod_{i \in \Delta} p_i$. This operation removes the primes $p_i, i \in \Delta$, scales down the polynomial by a factor of q_{diff} , and rounds so as to keep $a \bmod \text{ptxtSpace}$ unchanged.

We provide some conversion routines to convert polynomials from coefficient-representation (NTL's ZZX format) to `DoubleCRT` and back, using the constructor

```
DoubleCRT(const ZZX&, const FHEcontext&, const IndexSet&);
```

and the conversion function `ZZX to_ZZX(const DoubleCRT&)`. We also provide translation routines between `SingleCRT` and `DoubleCRT`.

We support the usual set of arithmetic operations on `DoubleCRT` objects (e.g., addition, multiplication, etc.), always working in \mathbb{A}_q for some modulus q . We only implemented the “destructive” two-argument version of these operations, where one of the input arguments is modified to return the result. These arithmetic operations can only be applied to `DoubleCRT` objects relative to the same `FHEcontext`, else an error is raised.

On the other hand, the `DoubleCRT` class supports operations between objects with different `IndexSet`'s, offering two options to resolve the differences: Our arithmetic operations take a boolean flag `matchIndexSets`, when the flag is set to `true` (which is the default), the index-set of the result is the union of the index-sets of the two arguments. When `matchIndexSets=false` then the index-set of the result is the same as the index-set of `*this`, i.e., the argument that will contain the result when the operation ends. The option `matchIndexSets=true` is slower, since it may require adding primes to the two arguments. Below is a list of the arithmetic routines that we implemented:

```
DoubleCRT& Negate(const DoubleCRT& other); // *this = -other
DoubleCRT& Negate();                      // *this = -*this;
```

```
DoubleCRT& operator+=(const DoubleCRT &other); // Addition
DoubleCRT& operator+=(const ZZX &poly); // expensive
DoubleCRT& operator+=(const ZZ &num);
DoubleCRT& operator+=(long num);
```

```
DoubleCRT& operator-=(const DoubleCRT &other); // Subtraction
DoubleCRT& operator-=(const ZZX &poly); // expensive
```

```

DoubleCRT& operator==(const ZZ &num);
DoubleCRT& operator==(long num);

// These are the prefix versions, ++dcrt and --dcrt.
DoubleCRT& operator++();
DoubleCRT& operator--();

// Postfix versions (return type is void, it is offered just for style)
void operator++(int);
void operator--(int);

DoubleCRT& operator*=(const DoubleCRT &other); // Multiplication
DoubleCRT& operator*=(const ZZx &poly); // expensive
DoubleCRT& operator*=(const ZZ &num);
DoubleCRT& operator*=(long num);

// Procedural equivalents, providing also the matchIndexSets flag

void Add(const DoubleCRT &other, bool matchIndexSets=true);
void Sub(const DoubleCRT &other, bool matchIndexSets=true);
void Mul(const DoubleCRT &other, bool matchIndexSets=true);

DoubleCRT& operator/=(const ZZ &num); // Division by constant
DoubleCRT& operator/=(long num);

void Exp(long k); // Small-exponent polynomial exponentiation

// Automorphism F(X) --> F(X^k) (with gcd(k,m)==1)
void automorph(long k);
DoubleCRT& operator>>=(long k);

```

We also provide methods for choosing at random polynomials in DoubleCRT format, as follows:

```

void randomize(const ZZ* seed=NULL);
    Fills each row  $i \in \text{getIndexSet}()$  with random integers modulo  $p_i$ . This procedure uses the
    NTL PRG, setting the seed to the seed argument if it is non-NULL, and using the current
    PRG state of NTL otherwise.

void sampleSmall();
    Draws a random polynomial with coefficients  $-1, 0, 1$ , and converts it to DoubleCRT format.
    Each coefficient is chosen as 0 with probability  $1/2$ , and as  $\pm 1$  with probability  $1/4$  each.

void sampleHWt(long weight);
    Draws a random polynomial with coefficients  $-1, 0, 1$ , and converts it to DoubleCRT format.
    The polynomial is chosen at random subject to the condition that all but weight of its
    coefficients are zero, and the non-zero coefficients are random in  $\pm 1$ .

```

```
void sampleGaussian(double stdev=3.2);
    Draws a random polynomial with coefficients  $-1, 0, 1$ , and converts it to DoubleCRT format.
    Each coefficient is chosen at random from a Gaussian distribution with zero mean and variance
     $\text{stdev}^2$ , rounded to an integer.
```

In addition to the above, we also provide the following methods:

```
DoubleCRT& SetZero(); // set to the constant zero
DoubleCRT& SetOne(); // set to the constant one
const FHEcontext& getContext() const; // access to context
const IndexSet& getIndexSet() const; // the current set of primes

void breakIntoDigits(vector<DoubleCRT>&, long) const; // used in key-switching
```

The method `breakIntoDigits` above is described in Section 3.1.6, where we discuss key-switching.

The SingleCRT class. `SingleCRT` is a helper class, used to gain some efficiency in expensive `DoubleCRT` operations. A `SingleCRT` object is also defined relative to a fixed `FHEcontext` and a dynamic subset S of the small primes. This `SingleCRT` object holds an `IndexMap` of polynomials (in NTL’s `ZZX` format), where the i ’th polynomial contains the coefficients modulo the i th small prime in our list.

Although `SingleCRT` and `DoubleCRT` objects can interact in principle, translation back and forth are expensive since they involve FFT (or inverse FFT) modulo each of the primes. Hence support for interaction between them is limited to explicit conversions.

3 The Crypto Layer

The third layer of our library contains the implementation of the actual BGV homomorphic cryptosystem, supporting homomorphic operations on the “native plaintext space” of polynomials in \mathbb{A}_2 (or more generally polynomials in \mathbb{A}_{2^r} for some parameter r). We partitioned this layer (somewhat arbitrarily) into the `Ctxt` module that implements ciphertexts and ciphertext arithmetic, the `FHE` module that implements the public and secret keys, and the key-switching matrices, and a helper `KeySwitching` module that implements some common strategies for deciding what key-switching matrices to generate. Two high-level design choices that we made in this layer is to implement ciphertexts as arbitrary-length vectors of polynomials, and to allow more than one secret-key per instance of the system. These two choices are described in more details in Sections 3.1 and 3.2 below, respectively.

3.1 The Ctxt module: Ciphertexts and homomorphic operations

Recall that in the BGV cryptosystem, a “canonical” ciphertext relative to secret key $\mathfrak{s} \in \mathbb{A}$ is a vector of two polynomials $(\mathfrak{c}_0, \mathfrak{c}_1) \in \mathbb{A}_q^2$ (for the “current modulus” q), such that $\mathfrak{m} = [\mathfrak{c}_0 + \mathfrak{c}_1 \mathfrak{s}]_q$ is a polynomial with small coefficients, and the plaintext that is encrypted by this ciphertext is the binary polynomial $[\mathfrak{m}]_2 \in \mathbb{A}_2$. However the library has to deal also with “non-canonical” ciphertexts: for example when multiplying two ciphertexts as above we get a vector of three polynomials $(\mathfrak{c}_0, \mathfrak{c}_1, \mathfrak{c}_2)$, which is encrypted by setting $\mathfrak{m} = [\mathfrak{c}_0 + \mathfrak{c}_1 \mathfrak{s} + \mathfrak{c}_2 \mathfrak{s}^2]_q$ and outputting $[\mathfrak{m}]_2$. Also, after a

homomorphic automorphism operation we get a two-polynomial ciphertext (c_0, c_1) but relative to the key $s' = \kappa(s)$ (where κ is the same automorphism that we applied to the ciphertext, namely $s'(X) = s(X^t)$ for some $t \in \mathbb{Z}_m^*$).

To support all of these options, a ciphertext in our library consists of an arbitrary-length vector of “ciphertext parts”, where each part is a polynomial, and each part contains a “handle” that points to the secret-key that this part should be multiply by during decryption. Handles, parts, and ciphertexts are implemented using the classes `SKHandle`, `CtxtPart`, and `Ctxt`, respectively.

3.1.1 The `SKHandle` class

An object of the `SKHandle` class “points” to one particular secret-key polynomial, that should multiply one ciphertext-part during decryption. Recall that we allow multiple secret keys per instance of the cryptosystem, and that we may need to reference powers of these secret keys (e.g. s^2 after multiplication) or polynomials of the form $s(X^t)$ (after automorphism). The general form of these secret-key polynomials is therefore $s_i^r(X^t)$, where s_i is one of the secret keys associated with this instance, r is the power of that secret key, and t is the automorphism that we applied to it. To uniquely identify a single secret-key polynomial that should be used upon decryption, we should therefore keep the three integers (i, r, t) .

Accordingly, a `SKHandle` object has three integer data members, `powerOfS`, `powerOfX`, and `secretKeyID`. It is considered a reference to the constant polynomial 1 whenever `powerOfS` = 0, irrespective of the other two values. Also, we say that a `SKHandle` object points to a *base secret key* if it has `powerOfS` = `powerOfX` = 1.

Observe that when multiplying two ciphertext parts, we get a new ciphertext part that should be multiplied upon decryption by the product of the two secret-key polynomials. This gives us the following set of rules for multiplying `SKHandle` objects (i.e., computing the handle of the resulting ciphertext-part after multiplication). Let $\{\text{powerOfS}, \text{powerOfX}, \text{secretKeyID}\}$ and $\{\text{powerOfS}', \text{powerOfX}', \text{secretKeyID}'\}$ be two handles to be multiplied, then we have the following rules:

- If one of the `SKHandle` objects points to the constant 1, then the result is equal to the other one.
- If neither points to one, then we must have `secretKeyID` = `secretKeyID'` and `powerOfX` = `powerOfX'`, otherwise we cannot multiply. If we do have these two equalities, then the result will also have the same $t = \text{powerOfX}$ and $i = \text{secretKeyID}$, and it will have $r = \text{powerOfS} + \text{powerOfS}'$.

The methods provided by the `SKHandle` class are the following:

```
SKHandle(long powerS=0, long powerX=1, long sKeyID=0); // constructor
long getPowerOfS() const;    // returns powerOfS;
long getPowerOfX() const;    // returns powerOfX;
long getSecretKeyID() const; // return secretKeyID;

void setBase(); // set to point to a base secret key
void setOne();  // set to point to the constant 1
bool isBase() const; // does it point to base?
```

```

bool isOne() const; // does it point to 1?

bool operator==(const SKHandle& other) const;
bool operator!=(const SKHandle& other) const;

bool mul(const SKHandle& a, const SKHandle& b); // multiply the handles
// result returned in *this, returns true if handles can be multiplied

```

3.1.2 The CtxtPart class

A ciphertext-part is a polynomial with a handle (that “points” to a secret-key polynomial). Accordingly, the class `CtxtPart` is derived from `DoubleCRT`, and includes an additional data member of type `SKHandle`. This class does not provide any methods beyond the ones that are provided by the base class `DoubleCRT`, except for access to the secret-key handle (and constructors that initialize it).

3.1.3 The Ctxt class

A `Ctxt` object is always defined relative to a fixed public key and context, both must be supplied to the constructor and are fixed thereafter. As described above, a ciphertext contains a vector of parts, each part with its own handle. This type of representation is quite flexible, for example you can in principle add ciphertexts that are defined with respect to different keys, as follows:

- For parts of the two ciphertexts that point to the same secret-key polynomial (i.e., have the same handle), you just add the two `DoubleCRT` polynomials.
- Parts in one ciphertext that do not have counter-part in the other ciphertext will just be included in the result intact.

For example, suppose that you wanted to add the following two ciphertexts. one “canonical” and the other after an automorphism $X \mapsto X^3$:

$$\begin{aligned}
\vec{c} &= (\mathbf{c}_0[i=0, r=0, t=0], \mathbf{c}_1[i=0, r=1, t=1]) \\
\text{and } \vec{c}' &= (\mathbf{c}'_0[i=0, r=0, t=0], \mathbf{c}'_3[i=0, r=1, t=3]).
\end{aligned}$$

Adding these ciphertexts, we obtain a three-part ciphertext,

$$\vec{c} + \vec{c}' = ((\mathbf{c}_0 + \mathbf{c}'_0)[i=0, r=0, t=0], \mathbf{c}_1[i=0, r=1, t=1], \mathbf{c}'_3[i=0, r=1, t=3]).$$

Similarly, we also have flexibility in multiplying ciphertexts using a tensor product, as long as all the pairwise handles of all the parts can be multiplied according to the rules from Section 3.1.1 above.

The `Ctxt` class therefore contains a data member `vector<CtxtPart> parts` that keeps all of the ciphertext-parts. By convention, the first part, `parts[0]`, always has a handle pointing to the constant polynomial 1. Also, we maintain the invariant that all the `DoubleCRT` objects in the parts of a ciphertext are defined relative to the same subset of primes, and the `IndexSet` for this subset is accessible as `ctxt.getPrimeSet()`. (The current BGV modulus for this ciphertext can be computed as $q = \text{ctxt.getContext().productOfPrimes}(\text{ctxt.getPrimeSet}())$.)

For reasons that will be discussed when we describe modulus-switching in Section 3.1.5, we maintain the invariant that a ciphertext relative to current modulus q and plaintext space p has an extra factor of $q \bmod p$. Namely, when we decrypt the ciphertext vector \vec{c} using the secret-key vector \vec{s} , we compute $\mathbf{m} = [\langle \vec{s}, \vec{c} \rangle]_p$ and then output the plaintext $m = [q^{-1} \cdot \mathbf{m}]_p$ (rather than just $m = [\mathbf{m}]_p$). Note that this has no effect when the plaintext space is $p = 2$, since $q^{-1} \bmod p = 1$ in this case.

The basic operations that we can apply to ciphertexts (beyond encryption and decryption) are addition, multiplication, addition and multiplication by constants, automorphism, key-switching and modulus switching. These operations are described in more details later in this section.

3.1.4 Noise estimate

In addition to the vector of parts, a ciphertext contains also a heuristic estimate of the “noise variance”, kept in the `noiseVar` data member: Consider the polynomial $\mathbf{m} = [\langle \vec{c}, \vec{s} \rangle]_q$ that we obtain during decryption (before the reduction modulo 2). Thinking of the ciphertext \vec{c} and the secret key \vec{s} as random variables, this makes also \mathbf{m} a random variable. The data members `noiseVar` is intended as an estimate of the second moment of the random variable $\mathbf{m}(\tau_m)$, where $\tau_m = e^{2\pi i/m}$ is the principal complex primitive m -th root of unity. Namely, we have $\text{noiseVar} \approx \mathbb{E}[|\mathbf{m}(\tau_m)|^2] = \mathbb{E}[\mathbf{m}(\tau_m) \cdot \overline{\mathbf{m}(\tau_m)}]$, with $\overline{\mathbf{m}(\tau_m)}$ the complex conjugate of $\mathbf{m}(\tau_m)$.

The reason that we keep an estimate of this second moment, is that it gives a convenient handle on the l_2 canonical embedding norm of \mathbf{m} , which is how we measure the noise magnitude in the ciphertext. Heuristically, the random variables $\mathbf{m}(\tau_m^j)$ for all $j \in \mathbb{Z}_m^*$ behave as if they are identically distributed, hence the expected squared l_2 norm of the canonical embedding of \mathbf{m} is

$$\mathbb{E}[(\|\mathbf{m}\|_2^{\text{canon}})^2] = \sum_{j \in \mathbb{Z}_m^*} \mathbb{E}[|\mathbf{m}(\tau_m^j)|^2] \approx \phi(m) \cdot \mathbb{E}[|\mathbf{m}(\tau_m)|^2] \approx \phi(m) \cdot \text{noiseVar}.$$

As the l_2 norm of the canonical embedding of \mathbf{m} is larger by a $\sqrt{\phi(m)}$ factor than the l_2 norm of its coefficient vector, we therefore use the condition $\sqrt{\text{noiseVar}} \geq q/2$ (with q the current BGV modulus) as our decryption-error condition. The library never checks this condition during the computation, but it provides a method `ctxt.isCorrect()` that the application can use to check for it. The library does use the noise estimate when deciding what primes to add/remove during modulus switching, see description of the `MultiplyBy` method below.

Recalling that the j 'th ciphertext part has a handle pointing to some $\mathfrak{s}_j^{r_j}(X^{t_j})$, we have that $\mathbf{m} = [\langle \vec{c}, \vec{s} \rangle]_q = [\sum_j \mathbf{c}_j \mathfrak{s}_j^{r_j}]_q$. A valid ciphertext vector in the BGV cryptosystem can always be written as $\vec{c} = \vec{r} + \vec{e}$ such that \vec{r} is some masking vector satisfying $[\langle \vec{r}, \vec{s} \rangle]_q = 0$ and $\vec{e} = (\mathbf{e}_1, \mathbf{e}_2, \dots)$ is such that $\|\mathbf{e}_j \cdot \mathfrak{s}_j^{r_j}(X^{t_j})\| \ll q$. We therefore have $\mathbf{m} = [\langle \vec{c}, \vec{s} \rangle]_q = \sum_j \mathbf{e}_j \mathfrak{s}_j^{r_j}$, and under the heuristic assumption that the “error terms” \mathbf{e}_j are independent of the keys we get $\mathbb{E}[|\mathbf{m}(\tau_m)|^2] = \sum_j \mathbb{E}[|\mathbf{e}_j(\tau_m)|^2] \cdot \mathbb{E}[|\mathfrak{s}_j(\tau_m^{t_j})^{r_j}|^2] \approx \sum_j \mathbb{E}[|\mathbf{e}_j(\tau_m)|^2] \cdot \mathbb{E}[|\mathfrak{s}_j(\tau_m)^{r_j}|^2]$. The terms $\mathbb{E}[|\mathbf{e}_j(\tau_m)|^2]$ depend on the particular error polynomials that arise during the computation, and will be described when we discuss the specific operations. For the secret-key terms we use the estimate

$$\mathbb{E}[|\mathfrak{s}(\tau_m)^r|^2] \approx r! \cdot H^r,$$

where H is the Hamming-weight of the secret-key polynomial \mathfrak{s} . For $r = 1$, it is easy to see that $\mathbb{E}[|\mathfrak{s}(\tau_m)|^2] = H$: Indeed, for every particular choice of the H nonzero coefficients of \mathfrak{s} , the random

variable $\mathfrak{s}(\tau_m)$ (defined over the choice of each of these coefficients as ± 1) is a zero-mean complex random variable with variance exactly H (since it is a sum of exactly H random variables, each obtained by multiplying a uniform ± 1 by a complex constant of magnitude 1). For $r > 1$, it is clear that $\mathbb{E}[|\mathfrak{s}(\tau_m)|^r] \geq \mathbb{E}[|\mathfrak{s}(\tau_m)|^2]^r = H^r$, but the factor of $r!$ may not be clear. We obtained that factor experimentally for the most part, by generating many polynomials \mathfrak{s} of some given Hamming weight and checking the magnitude of $\mathfrak{s}(\tau_m)$. Then we validated this experimental result the case $r = 2$ (which is the most common case when using our library), as described in the appendix. The rules that we use for computing and updating the data member `noiseVar` during the computation, as described below.

Encryption. For a fresh ciphertext, encrypted using the public encryption key, we have `noiseVar` = $\sigma^2(1 + \phi(m)^2/2 + \phi(m)(H + 1))$, where σ^2 is the variance in our RLWE instances, and H is the Hamming weight of the first secret key.

When the plaintext space modulus is $p > 2$, that quantity is larger by a factor of p^2 . See Section 3.2.2 for the reason for these expressions.

Modulus-switching. The noise magnitude in the ciphertexts scales up as we add primes to the prime-set, while modulus-switching down involves both scaling down and adding some term (corresponding to the rounding errors for modulus-switching). Namely, when adding more primes to the prime-set we scale up the noise estimate as `noiseVar'` = `noiseVar` $\cdot \Delta^2$, with Δ the product of the added primes.

When removing primes from the prime-set we scale down and add an extra term, setting `noiseVar'` = `noiseVar`/ Δ^2 + `addedNoise`, where the added-noise term is computed as follows: We go over all the parts in the ciphertext, and consider their handles. For any part j with a handle that points to $\mathfrak{s}_j^{r_j}(X^{t_j})$, where \mathfrak{s}_j is a secret-key polynomial whose coefficient vector has Hamming-weight H_j , we add a term $(p^2/12) \cdot \phi(m) \cdot (r_j)! \cdot H_j^{r_j}$. Namely, when modulus-switching down we set

$$\text{noiseVar}' = \text{noiseVar}/\Delta^2 + \sum_j \frac{p^2}{12} \cdot \phi(m) \cdot (r_j)! \cdot H_j^{r_j}.$$

See Section 3.1.5 for the reason for this expression.

Re-linearization/key-switching. When key-switching a ciphertext, we modulus-switch down to remove all the “special primes” from the prime-set of the ciphertext if needed (cf. Section 2.7). Then, the key-switching operation itself has the side-effect of adding these “special primes” back. These two modulus-switching operations have the effect of scaling the noise down, then back up, with the added noise term as above. Then add yet another noise term as follows:

The key-switching operation involves breaking the ciphertext into some number n' of “digits” (see Section 3.1.6). For each digit i of size D_i and every ciphertext-part that we need to switch (i.e., one that does not already point to 1 or a base secret key), we add a noise-term $\phi(m)\sigma^2 \cdot p^2 \cdot D_i^2/4$, where σ^2 is the variance in our RLWE instances. Namely, if we need to switch k parts and if `noiseVar'` is the noise estimate after the modulus-switching down and up as above, then our final noise estimate after key-switching is

$$\text{noiseVar}'' = \text{noiseVar}' + k \cdot \phi(m)\sigma^2 \cdot p^2 \cdot \sum_{i=1}^{n'} D_i^2/4$$

where D_i is the size of the i 'th digit. See Section 3.1.6 for more details.

addConstant. We roughly add the size of the constant to our noise estimate. The calling application can either specify the size of the constant, or else we use the default value $\mathbf{sz} = \phi(m) \cdot (p/2)^2$. Recalling that when current modulus is q we need to scale up the constant by $q \bmod p$, we therefore set $\mathbf{noiseVar}' = \mathbf{noiseVar} + (q \bmod p)^2 \cdot \mathbf{sz}$.

multByConstant. We multiply our noise estimate by the size of the constant. Again, the calling application can either specify the size of the constant, or else we use the default value $\mathbf{sz} = \phi(m) \cdot (p/2)^2$. Then we set $\mathbf{noiseVar}' = \mathbf{noiseVar} \cdot \mathbf{sz}$.

Addition. We first add primes to the prime-set of the two arguments until they are both defined relative to the same prime-set (i.e. the union of the prime-sets of both arguments). Then we just add the noise estimates of the two arguments, namely $\mathbf{noiseVar}' = \mathbf{noiseVar} + \mathbf{other.noiseVar}$.

Multiplication. We first remove primes from the prime-set of the two arguments until they are both defined relative to the same prime-set (i.e. the intersection of the prime-sets of both arguments). Then the noise estimate is set to the product of the noise estimates of the two arguments, multiplied by an additional factor which is computed as follows: Let r_1 be the highest power of \mathfrak{s} (i.e., the `powerOfS` value) in all the handles in the first ciphertext, and similarly let r_2 be the highest power of \mathfrak{s} in all the handles in the second ciphertext, then the extra factor is $\binom{r_1+r_2}{r_1}$. Namely, we have $\mathbf{noiseVar}' = \mathbf{noiseVar} \cdot \mathbf{other.noiseVar} \cdot \binom{r_1+r_2}{r_1}$. (In particular if the two arguments are canonical ciphertexts then the extra factor is $\binom{2}{1} = 2$.) See Section 3.1.7 for more details.

Automorphism. The noise estimate does not change by an automorphism operation.

3.1.5 Modulus-switching operations

Our library supports modulus-switching operations, both adding and removing small primes from the current prime-set of a ciphertext. In fact, our decision to include an extra factor of $(q \bmod p)$ in a ciphertext relative to current modulus q and plaintext-space modulus p , is mainly intended to somewhat simplify these operations.

To add primes, we just apply the operation `addPrimesAndScale` to all the ciphertext parts (which are polynomials in Double-CRT format). This has the effect of multiplying the ciphertext by the product of the added primes, which we denote here by Δ , and we recall that this operation is relatively cheap (as it involves no FFTs or CRTs, cf. Section 2.8). Denote the current modulus before the modulus-UP transformation by q , and the current modulus after the transformation by $q' = q \cdot \Delta$. If before the transformation we have $[\langle \vec{c}, \vec{s} \rangle]_q = \mathbf{m}$, then after this transformation we have $\langle \vec{c}', \vec{s} \rangle = \langle \Delta \cdot \vec{c}, \vec{s} \rangle = \Delta \cdot \langle \vec{c}, \vec{s} \rangle$, and therefore $[\langle \vec{c}', \vec{s} \rangle]_{q \cdot \Delta} = \Delta \cdot \mathbf{m}$. This means that if before the transformation we had by our invariant $[\langle \vec{c}, \vec{s} \rangle]_q = \mathbf{m} \equiv q \cdot m \pmod{p}$, then after the transformation we have $[\langle \vec{c}', \vec{s} \rangle]_{q'} = \Delta \cdot \mathbf{m} \equiv q' \cdot m \pmod{p}$, as needed.

For a modulus-DOWN operation (i.e., removing primes) from the current modulus q to the smallest modulus q' , we need to scale the ciphertext \vec{c} down by a factor of $\Delta = q/q'$ (thus getting a fractional ciphertext), then round appropriately to get back an integer ciphertext. Using our invariant about the extra factor of $(q \bmod p)$ in a ciphertext relative to modulus q (and plaintext

space modulus p), we need to convert \vec{c} into another ciphertext vector \vec{c}' satisfying (a) $(q')^{-1}\vec{c}' \equiv q^{-1}\vec{c} \pmod{p}$, and (b) the “rounding error term” $\epsilon \stackrel{\text{def}}{=} \vec{c}' - (q'/q)\vec{c}$ is small. As described in [5], we apply the following optimized procedure:

1. Let $\vec{\delta} = \vec{c} \bmod \Delta$,
2. Add or subtract multiples of Δ from the coefficients in $\vec{\delta}$ until it is divisible by p ,
3. Set $\vec{c}^* = \vec{c} - \vec{\delta}$, // \vec{c}^* divisible by Δ , and $\vec{c}^* \equiv \vec{c} \pmod{p}$
4. Output $\vec{c}' = \vec{c}^*/\Delta$.

An argument similar to the proof of [2, Lemma 4] shows that if before the transformation we had $\mathbf{m} = [\langle \vec{c}, \vec{s} \rangle]_q \equiv q \cdot m \pmod{p}$, then after the transformation we have $\mathbf{m}' = [\langle \vec{c}', \vec{s} \rangle]_{q'} \equiv q' \cdot m \pmod{p}$, as needed. (The difference from [2, Lemma 4] is that we do not assume that $q, q' \equiv 1 \pmod{p}$.)

Considering the noise magnitude, we can write $\vec{c}' = \vec{c}/\Delta + \vec{\epsilon}$ where $\vec{\epsilon}$ is the rounding error (i.e., the terms that are added in Step 2 above, divided by Δ). The noise polynomial is thus scaled down by a Δ factor, then increased by the additive term $a \stackrel{\text{def}}{=} \langle \vec{\epsilon}, \vec{s} \rangle = \sum_j \epsilon_j(X) \cdot \mathfrak{s}_j^{r_j}(X^{t_j})$ (with $a \in \mathbb{A}$). We make the heuristic assumption that the coefficients in all the ϵ_j ’s behave as if they are chosen uniformly in the interval $[-p/2, p/2)$. Under this assumption, we have

$$\mathbb{E} \left[|\epsilon_j(\tau_m)|^2 \right] = \phi(m) \cdot p^2/12,$$

since the variance of a uniform random variable in $[-p/2, p/2)$ is $p^2/12$, and $\epsilon_j(\tau_m)$ is a sum of $\phi(m)$ such variables, scaled by different magnitude-1 complex constants. Assuming heuristically that the ϵ_j ’s are independent of the public key, we have

$$\mathbb{E} \left[|a(\tau_m)|^2 \right] = \sum_j \mathbb{E} \left[|\epsilon_j(\rho_m)|^2 \right] \cdot \mathbb{E} \left[\left| \mathfrak{s}_j^{r_j}(X^{t_j}) \right|^2 \right] \approx \sum_j (\phi(m) \cdot p^2/12) \cdot (r_j)! \cdot H_j^{r_j},$$

where p is the plaintext-space modulus, H_j is the Hamming weight of the secret key for the j ’th part, and r_j is the power of that secret key.

3.1.6 Key-switching/re-linearization

The re-linearization operation ensures that all the ciphertext parts have handles that point to either the constant 1 or a base secret-key: Any ciphertext part j with a handle pointing to $\mathfrak{s}_j^{r_j}(X^{t_j})$ with either $r_j > 1$ or $r_j = 1$ and $t_j > 1$, is replace by two adding two parts, one that points to 1 and the other than points to $\mathfrak{s}_j(X)$, using some key-switching matrices from the public key. Also, a side-effect of re-linearization is that we add all the “special primes” to the prime-set of the resulting ciphertext.

To explain the re-linearization procedure, we begin by recalling that the “ciphertext primes” that define our moduli-chain are partitioned into some number $n \geq 1$ of “digits”, of roughly equal size. (For example, say that we have 15 small primes in the chain and we partition them to three digits, then we may take the first five primes to be the first digit, the next five primes to be the second, and the last five primes to be the third.) The size of a digit is the product of all the primes that are associated with it, and below we denote by D_i the size of the i ’th digit.

When key-switching a ciphertext \vec{c} using $n > 1$ digits, we begin by breaking \vec{c} into a collection of (at most) n lower-norm ciphertexts \vec{c}_i . First we remove all the special primes from the prime-set of the ciphertext by modulus-DOWN, if needed. Then we determine the smallest n' such that the product of the first n' digits exceeds the current modulus q , and then we set

1. $\vec{d}_1 := \vec{c}$
2. For $i = 1$ to n' do:
3. $\vec{c}_i := \vec{d}_i \bmod D_i$ // $|\vec{c}_i| < D_i/2$
4. $\vec{d}_{i+1} := (\vec{d}_i - \vec{c}_i)/D_i$ // $(\vec{d}_i - \vec{c}_i)$ divisible by D_i

Note that $\vec{c} = \sum_{i=1}^{n'} \vec{c}_i \cdot \prod_{j<i} D_j$, and also since $\|\vec{c}\|_\infty \leq q/2 \leq (\prod_i D_i)/2$, then it follows that $\|\vec{c}_i\|_\infty \leq D_i/2$ for all i . Below we assume that the current modulus q is equal to the product of the first n' digits. (The case where $q < \prod_i D_i$ is very similar, but requires somewhat more complicated notations, in that case we just remove primes from the last digit until the product is equal to q .)

Consider now one particular ciphertext-part \mathbf{c}_j in \vec{c} , with a handle that points to some $\mathbf{s}'_j(X) = \mathbf{s}_j^{r_j j}(X^{t_j})$, with either $r_j > 1$ or $r_j = 1$ and $t_j > 1$. Let us denote by $\mathbf{c}_{i,j}$ the ciphertext-parts corresponding to \mathbf{c}_j in the low-norm ciphertexts \vec{c}_i . That is, we have $\mathbf{c}_j = \sum_{i=1}^{n'} \mathbf{c}_{i,j} \cdot \prod_{j<i} D_i$, and also $\|\mathbf{c}_{i,j}\|_\infty \leq D_i/2$ for all i . Moreover we need to have in the public-key a key-switching matrix for that handle, $W = W[\mathbf{s}'_j \Rightarrow \mathbf{s}_j]$. This W is a $2 \times n$ matrix of polynomials in Double-CRT format, defined relative to the product of all the small primes in our chain (special or otherwise). Below we denote the product of all these small primes by Q . The i 'th column in the matrix encrypts the “plaintext polynomial” $\mathbf{s}'_j \cdot \prod_{j<i} D_i$ under the key \mathbf{s}_j , namely a vector $(\mathbf{a}_i, \mathbf{b}_i)^T \in A_Q^2$ such that $[\mathbf{b}_i + \mathbf{a}_i \mathbf{s}_j]_Q = (\prod_{j<i} D_i) \cdot \mathbf{s}'_j + p \cdot \mathbf{e}_i$, for a small polynomial \mathbf{e}_i (and the plaintext-space modulus p). Moreover, as long as $(\prod_{j<i} D_i) \cdot \mathbf{s}'_j + p \cdot \mathbf{e}_i$ is short enough, the same equality holds also modulo smaller moduli that divide Q . In particular, denoting the product of the “special primes” by Q^* and the product of the n' digits that we use by q , then for all $i \leq n'$ we have $\|(\prod_{j<i} D_i) \cdot \mathbf{s}'_j + p \cdot \mathbf{e}_i\|_\infty < qQ^*/2$, and therefore

$$[\mathbf{b}_i + \mathbf{a}_i \mathbf{s}_j]_{qQ^*} = \left(\prod_{j<i} D_i \right) \cdot \mathbf{s}'_j + p \cdot \mathbf{e}_i.$$

We therefore reduce the key-switching matrix modulo qQ^* , and add the small primes corresponding to qQ^* to all the $\mathbf{c}_{i,j}$'s. We replace \mathbf{c}_j by ciphertext-parts that point to 1 and base, by multiplying the n' -vector $\tilde{c}_j = (\mathbf{c}_{1,j}, \dots, \mathbf{c}_{n',j})^T$ by (the first n' columns of) the key-switching matrix W , setting

$$(\mathbf{c}''_j, \mathbf{c}'_j)^T := [W[1 : n'] \times \tilde{c}_j]_{qQ^*} = \left[\sum_{i=1}^{n'} (\mathbf{a}_i, \mathbf{b}_i)^T \cdot \mathbf{c}_{i,j} \right]_{qQ^*}.$$

It is not hard to see that for these two new ciphertext-parts we have:

$$\begin{aligned} \mathbf{c}''_j + \mathbf{c}'_j \mathbf{s}_j &= \sum_{i=1}^{n'} (\mathbf{b}_i + \mathbf{a}_i \mathbf{s}_j) \cdot \mathbf{c}_{i,j} = \sum_{i=1}^{n'} \left(\left(\prod_{j<i} D_i \right) \cdot \mathbf{s}'_j + p \cdot \mathbf{e}_i \right) \cdot \mathbf{c}_{i,j} \\ &= \left(\sum_{i=1}^{n'} \left(\prod_{j<i} D_i \right) \mathbf{c}_{i,j} \right) \mathbf{s}'_j + p \cdot \sum_{i=1}^{n'} \mathbf{e}_i \mathbf{c}_{i,j} = \mathbf{c}_j \mathbf{s}'_j + p \sum_{i=1}^{n'} \mathbf{e}_i \mathbf{c}_{i,j} \pmod{qQ^*} \end{aligned}$$

Replacing all the parts \mathbf{c}_j by pairs $(\mathbf{c}''_j, \mathbf{c}'_j)^T$ as above (and adding up all the parts that point to 1, as well as all the parts that point to the base \mathbf{s}_j), we thus get a canonical ciphertext $\vec{c}' = (\tilde{c}_2, \tilde{c}_1)^T$,

with $\tilde{\mathbf{c}}_2 = [\sum_j \mathbf{c}_j'']_{qQ^*}$ and $\tilde{\mathbf{c}}_1 = [\sum_j \mathbf{c}_j']_{qQ^*}$, and we have

$$\tilde{\mathbf{c}}_2 + \tilde{\mathbf{c}}_1 \mathbf{s}_j = \left(\sum_j \mathbf{c}_j \mathbf{s}_j' \right) + p \left(\sum_{i,j} \mathbf{e}_i \mathbf{c}_{i,j} \right) = \mathbf{m} + p \left(\sum_{i,j} \mathbf{e}_i \mathbf{c}_{i,j} \right) \pmod{qQ^*}.$$

Hence, as long as the additive term $p(\sum_{i,j} \mathbf{e}_i \mathbf{c}_{i,j})$ is small enough, decrypting the new ciphertext yields the same plaintext value modulo p as decrypting the original ciphertext \vec{c} .

In terms of noise magnitude, we first scale up the noise by a factor of Q^* when adding all the special primes, and then add the extra noise term $p \cdot \sum_{i,j} \mathbf{e}_i \mathbf{c}_{i,j}$. Since the $\mathbf{c}_{i,j}$'s have coefficients of magnitude at most $D_i/2$ and the polynomials \mathbf{e}_i are RLWE error terms with zero-mean coefficients and variance σ^2 , then the second moment of $\mathbf{e}_i(\tau_m) \cdot \mathbf{c}_{i,j}(\tau_m)$ is no more than $\phi(m)\sigma^2 \cdot D_i^2/4$. Thus for every ciphertext part that we need to switch (i.e. that has a handle that points to something other than 1 or base), we add a term of $\phi(m)\sigma^2 \cdot p^2 \cdot D_i^2/4$. Therefore, if our noise estimate after the scale-up is $\text{noiseVar}'$ and we need to switch k

$$\text{noiseVar}'' = \text{noiseVar}' + k \cdot \phi(m)\sigma^2 \cdot p^2 \cdot \sum_{i=1}^{n'} D_i^2/4$$

3.1.7 Native arithmetic operations

The native arithmetic operations that can be performed on ciphertexts are negation, addition/subtraction, multiplication, addition of constants, multiplication by constant, and automorphism. In our library we expose to the application both the operations in their “raw form” without any additional modulus- or key-switching, as well as some higher-level interfaces for multiplication and automorphisms that include also modulus- and key-switching.

Negation. The method `Ctxt::negate()` transforms an encryption of a polynomial $m \in \mathbb{A}_p$ to an encryption of $[-m]_p$, simply by negating all the ciphertext parts modulo the current modulus. (Of course this has an effect on the plaintext only when $p > 2$.) The noise estimate is unaffected.

Addition/subtraction. Both of these operations are implemented by the single method

```
void Ctxt::addCtxt(const Ctxt& other, bool negative=false);
```

depending on the `negative` boolean flag. For convenience, we provide the methods `Ctxt::operator+=` and `Ctxt::operator-=` that call `addCtxt` with the appropriate flag. A side effect of this operation is that the prime-set of `*this` is set to the union of the prime sets of both ciphertexts. After this scaling (if needed), every ciphertext-part in `other` that has a matching part in `*this` (i.e. a part with the same handle) is added to this matching part, and any part in `other` without a match is just appended to `*this`. We also add the noise estimate of both ciphertexts.

Constant addition. Implemented by the methods

```
void Ctxt::addConstant(const ZZx& poly, double size=0.0);
```

```
void Ctxt::addConstant(const DoubleCRT& poly, double size=0.0);
```

The constant is scaled by a factor $f = (q \bmod p)$, with q the current modulus and p the ciphertext modulus (to maintain our invariant that a ciphertext relative to q has this extra factor), then added to the part of `*this` that points to 1. The calling application can specify the size of the added

constant (i.e. $|\text{poly}(\tau_m)|^2$), or else we use the default value $\text{size} = \phi(m) \cdot (p/2)^2$, and this value (times f^2) is added to our noise estimate.

Multiplication by constant. Implemented by the methods

```
void Ctxt::multByConstant(const ZZx& poly, double size=0.0);
void Ctxt::multByConstant(const DoubleCRT& poly, double size=0.0);
```

All the parts of `*this` are multiplied by the constant, and the noise estimate is multiplied by the size of the constant. As before, the application can specify the size, or else we use the default value $\text{size} = \phi(m) \cdot (p/2)^2$.

Multiplication. “Raw” multiplication is implemented by

```
Ctxt& Ctxt::operator*=(const Ctxt& other);
```

If needed, we modulus-switch down to the intersection of the prime-sets of both arguments, then compute the tensor product of the two, namely the collection of all pairwise products of parts from `*this` and `other`.

The noise estimate of the result is the product of the noise estimates of the two arguments, times a factor which is computed as follows: Let r_1 be the highest power of \mathfrak{s} (i.e., the `powerOfS` value) in all the handles in `*this`, and similarly let r_2 be the highest power of \mathfrak{s} in all the handles `other`. The extra factor is then set as $\binom{r_1+r_2}{r_1}$. Namely, $\text{noiseVar}' = \text{noiseVar} \cdot \text{other.noiseVar} \cdot \binom{r_1+r_2}{r_1}$. The reason for the $\binom{r_1+r_2}{r_1}$ factor is that the ciphertext part in the result, obtained by multiplying the two parts with the highest `powerOfS` value, will have `powerOfS` value of the sum, $r = r_1 + r_2$. Recall from Section 3.1.4 that we estimate $\mathbb{E}[|\mathfrak{s}(\tau_m)^r|^2] \approx r! \cdot H^r$, where H is the Hamming weight of the coefficient-vector of \mathfrak{s} . Thus our noise estimate for the relevant part in `*this` is $r_1! \cdot H^{r_1}$ and the estimate for the part in `other` is $r_2! \cdot H^{r_2}$. To obtain the desired estimate of $(r_1 + r_2)! \cdot H^{r_1+r_2}$, we need to multiply the product of the estimates by the extra factor $\frac{(r_1+r_2)!}{r_1! \cdot r_2!} = \binom{r_1+r_2}{r_1}$.¹

Higher-level multiplication. We also provide the higher-level methods

```
void Ctxt::multiplyBy(const Ctxt& other);
void Ctxt::multiplyBy(const Ctxt& other1, const Ctxt& other2);
```

The first method multiplies two ciphertexts, it begins by removing primes from the two arguments down to a level where the rounding-error from modulus-switching is the dominating noise term (see `findBaseSet` below), then it calls the low-level routine to compute the tensor product, and finally it calls the `reLinearize` method to get back a canonical ciphertext.

The second method that multiplies three ciphertexts also begins by removing primes from the two arguments down to a level where the rounding-error from modulus-switching is the dominating noise term. Based on the prime-sets of the three ciphertexts it chooses an order to multiply them (so that ciphertexts at higher levels are multiplied first). Then it calls the tensor-product routine to multiply the three arguments in order, and then re-linearizes the result.

We also provide the two convenience methods `square` and `cube` that call the above two-argument and three-argument multiplication routines, respectively.

¹Although products of other pairs of parts may need a smaller factor, the parts with highest `powerOfS` value represent the largest contribution to the overall noise, hence we use this largest factor for everything.

Automorphism. “Raw” automorphism is implemented in the method

```
void Ctxt::automorph(long k);
```

For convenience we also provide `Ctxt& operator>>=(long k);` that does the same thing. These methods just apply the automorphism $X \mapsto X^k$ to every part of the current ciphertext, without changing the noise estimate, and multiply by k (modulo m) the `powerOfX` value in the handles of all these parts.

“Smart” Automorphism. Higher-level automorphism is implemented in the method

```
void Ctxt::smartAutomorph(long k);
```

The difference between `automorph` and `smartAutomorph` is that the latter ensures that the result can be re-linearized using key-switching matrices from the public key. Specifically, `smartAutomorph` breaks the automorphism $X \mapsto X^k$ into some number $t \geq 1$ of steps, $X \mapsto X^{k_i}$ for $i = 1, 2, \dots, t$, such that the public key contains key-switching matrices for re-linearizing all these steps (i.e. $W = W[\mathfrak{s}(X^{k_i}) \Rightarrow \mathfrak{s}(X)]$), and at the same time we have $\prod_{i=1}^t k_i = k \pmod{m}$. The method `smartAutomorph` then begin by re-linearizing its argument, then in every step it performs one of the automorphisms $X \mapsto X^{k_i}$ followed by re-linearization.

The decision of how to break each exponent k into a sequence of k_i ’s as above is done off line during key-generation, as described in Section 3.2.2. After this off-line computation, the public key contains a table that for each $k \in \mathbb{Z}_m^*$ indicates what is the first step to take when implementing the automorphism $X \mapsto X^k$. The `smartAutomorph` looks up the first step k_1 in that table, performs the automorphism $X \mapsto X^{k_1}$, then compute $k' = k/k_1 \pmod{m}$ and does another lookup in the table for the first step relative to k' , and so on.

3.1.8 More Ctxt methods

The `Ctxt` class also provide the following utility methods:

```
void clear();
```

 Removes all the parts and sets the noise estimate to zero.

```
xdouble modSwitchAddedNoiseVar() const;
```

 computes the added-noise from modulus-switching, namely it returns $\sum_j (\phi(m) \cdot p^2/12) \cdot (r_j)! \cdot H_j^{r_j}$ where H_j and r_j are respectively the Hamming weight of the secret key that the j ’th ciphertext-part points to, and the power of that secret key (i.e., the `powerOfS` value in the relevant handle).

```
void findBaseSet(IndexSet& s) const;
```

 Returns in `s` the largest prime-set such that modulus-switching to `s` would make `ctxt.modSwitchAddedNoiseVar` the most significant noise term. In other words, modulus-switching to `s` results in a significantly smaller noise than to any larger prime-set, but modulus-switching further down would not reduce the noise by much. When multiplying ciphertexts using the `multiplyBy` “high-level” methods, the ciphertexts are reduced to (the intersection of) their “base sets” levels before multiplying.

```
long getLevel() const;
```

 Returns the number of primes in the result of `findBaseSet`.

```
bool inCanonicalForm(long keyID=0) const;
```

 Returns *true* if this is a canonical ciphertexts, with only two parts: one that points to 1 and the other that points to the “base” secret key $\mathfrak{s}_i(X)$, (where $i = \text{keyId}$ is specified by the caller).

`bool isCorrect() const;` and `double log_of_ratio() const;` The method `isCorrect()` compares the noise estimate to the current modulus, and returns `true` if the noise estimate is less than half the modulus size. Specifically, if $\sqrt{\text{noiseVar}} < q/2$. The method `double log_of_ratio()` returns $\log(\text{noiseVar})/2 - \log(q)$.

Access methods. Read-only access the data members of a `Ctxt` object:

```
const FHEcontext& getContext() const;
const FHEPubKey& getPubKey() const;
const IndexSet& getPrimeSet() const;
const xdouble& getNoiseVar() const;
const long getPtTxtSpace() const; // the plaintext-space modulus
const long getKeyID() const;      // key-ID of the first part not pointing to 1
```

3.2 The FHE module: Keys and key-switching matrices

Recall that we made the high-level design choices to allow instances of the cryptosystem to have multiple secret keys. This decision was made to allow a *leveled* encryption system that does not rely on circular security, as well as to support switching to a different key for different purposes (which may be needed for bootstrapping, for example). However, we still view using just a single secret-key per instance (and relying on circular security) as the primary mode of using our library, and hence provided more facilities to support this mode than for the mode of using multiple keys. Regardless of how many secret keys we have per instance, there is always just a single public encryption key, for encryption relative to the first secret key. (The public key in our variant of the BGV cryptosystem is just a ciphertext, encrypting the constant 0.) In addition to this public encryption key, the public-key contains also key-switching matrices and some tables to help finding the right matrices to use in different settings. Ciphertexts relative to secret keys other than the first (if any), can only be generated using the key-switching matrices in the public key.

3.2.1 The KeySwitch class

This class implements key-switching matrices. As we described in Section 3.1.6, a key-switching matrix from \mathfrak{s}' to \mathfrak{s} , denoted $W[\mathfrak{s}' \Rightarrow \mathfrak{s}]$, is a $2 \times n$ matrix of polynomials from \mathbb{A}_Q , where Q is the product of all the small primes in our chain (both ciphertext-primes and special-primes). Recall that the ciphertext primes are partitioned into n digits, where we denote the product of primes corresponding the i 'th digit by D_i . Then the i 'th column of the matrix $W[\mathfrak{s}' \Rightarrow \mathfrak{s}]$ is a pair of elements $(\mathbf{a}_i, \mathbf{b}_i) \in \mathbb{A}_Q^2$ that satisfy

$$[\mathbf{b}_i + \mathbf{a}_i \cdot \mathfrak{s}]_Q = \left(\prod_{j < i} D_j \right) \cdot \mathfrak{s}' + p \cdot \mathbf{e}_i,$$

for a low-norm polynomial $\mathbf{e}_i \in \mathbb{A}_Q$. In more detail, we choose a low-norm polynomial $\mathbf{e}_i \in \mathbb{A}_Q$, where each coefficient of \mathbf{e}_i is chosen from a discrete Gaussian over the integers with variance σ^2 (with σ a parameter, by default $\sigma = 3.2$). Then we choose a random polynomial $\mathbf{a}_i \in_R \mathbb{A}_Q$ and set $\mathbf{b}_i := \left[\left(\prod_{j < i} D_j \right) \cdot \mathfrak{s}' + p \cdot \mathbf{e}_i - \mathbf{a}_i \cdot \mathfrak{s} \right]_Q$.

The matrix W is stored in a `KeySwitch` object in a space-efficient manner: instead of storing the random polynomials \mathbf{a}_i themselves, we only store a seed for a pseudorandom-generator, from which

all the \mathbf{a}_i 's are derived. The \mathbf{b}_i 's are stored explicitly, however. We note that this space-efficient representation requires that we assume hardness of our ring-LWE instances even when the seed for generating the random elements is known, but this seems like a reasonable assumption.

In our library, the source secret key \mathbf{s}' is of the form $\mathbf{s}' = \mathbf{s}_{i'}^r(X^t)$ (for some index i' and exponents r, t), but the target \mathbf{s} must be a “base” secret-key, i.e. $\mathbf{s} = \mathbf{s}_i(X)$ for some index i . The `KeySwitch` object stores in addition to the matrix W also a secret-key handle (r, t, i') that identifies the source secret key, as well as the index i of the target secret key.

The `KeySwitch` class provides a method `NumCols()` that returns the number of columns in the matrix W . We maintain the invariant that all the key-switching matrices that are defined relative to some context have the same number of columns, which is also equal to the number of digits that are specified in the context.

3.2.2 The `FHEPubKey` class

An `FHEPubKey` object is defined relative to a fixed `FHEcontext`, which must be supplied to the constructor and cannot be changed later. An `FHEPubKey` includes the public encryption key (which is a ciphertext of type `Ctxt`), a vector of key-switching matrices (of type `KeySwitch`), and another data structure (called `keySwitchMap`) that is meant to help finding the right key-switching matrices to use for automorphisms (see a more detailed description below). In addition, for every secret key in this instance, the `FHEPubKey` object stores the Hamming weight of that key, i.e., the number of non-zero coefficients of the secret-key polynomial. (This last piece of information is used to compute the estimated noise in a ciphertext.) The `FHEPubKey` class provides an encryption method, and various methods to find and access key-switching matrices.

`long Encrypt(Ctxt& ciphertext, const ZZ& plaintext, long ptxtSpace=0) const;` This method returns in `ciphertext` an encryption of the plaintext polynomial `plaintext`, relative to the plaintext-space modulus given in `ptxtSpace`. If the `ptxtSpace` parameter is not specified then we use the plaintext-space modulus from the public encryption key in this `FHEPubKey` object, and otherwise we use the greater common divisor (GCD) of the specified value and the one from the public encryption key. The current-modulus in the new fresh ciphertext is the product of all the ciphertext-primes in the context, which is the same as the current modulus in the public encryption key in this `FHEPubKey` object.

Let the public encryption key in the `FHEPubKey` object be denoted $\vec{c}^* = (\mathbf{c}_0^*, \mathbf{c}_1^*)$, let Q_{ct} be the product of all the ciphertext primes in the context, and let p be the plaintext-space modulus (namely the GCD of the parameter `ptxtSpace` and the plaintext-space modulus from the public encryption key). The `Encrypt` method chooses a random low-norm polynomial $\mathbf{r} \in \mathbb{A}_{Q_{\text{ct}}}$ with $-1/0/1$ coefficients, and low-norm error polynomials $\mathbf{e}_0, \mathbf{e}_1 \in \mathbb{A}_Q$, where each coefficient of \mathbf{e}_i 's is chosen from a discrete Gaussian over the integers with variance σ^2 (with σ a parameter, by default $\sigma = 3.2$). We then compute and return the canonical ciphertext

$$\vec{c} = (\mathbf{c}_0, \mathbf{c}_1) := \mathbf{r} \cdot (\mathbf{c}_0^*, \mathbf{c}_1^*) + p \cdot (\mathbf{e}_0, \mathbf{e}_1) + \text{plaintext}.$$

Note that since the public encryption key satisfies $[\mathbf{c}_0^* + \mathbf{s} \cdot \mathbf{c}_1^*]_{Q_{\text{ct}}} = p \cdot \mathbf{e}^*$ for some low-norm polynomial \mathbf{e}^* , then we have

$$[\mathbf{c}_0 + \mathbf{s} \cdot \mathbf{c}_1]_{Q_{\text{ct}}} = [\mathbf{r} \cdot (\mathbf{c}_0^* + \mathbf{s} \cdot \mathbf{c}_1^*) + p \cdot (\mathbf{e}_0 + \mathbf{s} \cdot \mathbf{e}_1) + \text{plaintext}]_{Q_{\text{ct}}} = p \cdot (\mathbf{e}_0 + \mathbf{s} \cdot \mathbf{e}_1 + \mathbf{r} \cdot \mathbf{e}^*) + \text{plaintext}.$$

For the noise estimate in the new ciphertext, we multiply the noise estimate in the public encryption key by the size of the low-norm \mathbf{r} , and add another term for the expression $\mathbf{a} = p \cdot (\mathbf{e}_0 + \mathbf{s} \cdot \mathbf{e}_1) + \text{plaintext}$. Specifically, the noise estimate in the public encryption key is `pubEncrKey.noiseVar` = $\phi(m)\sigma^2p^2$, the second moment of $\mathbf{r}(\tau_m)$ is $\phi(m)/2$, and the second moment of $a(\tau_m)$ is no more than $p^2(1 + \text{sigma}2^\phi(m)(H+1))$ with H the Hamming weight of the secret key \mathbf{s} . Hence the noise estimate in a freshly encrypted ciphertext is

$$\text{noiseVar} = p^2 \cdot (1 + \sigma^2\phi(m) \cdot (\phi(m)/2 + H + 1)).$$

The key-switching matrices. An `FHEPubKey` object keeps a list of all the key-switching matrices that were generated during key-generation in the data member `keySwitching` of type `vector<KeySwitch>`. As explained above, each key-switching matrix is of the form $W[\mathbf{s}_i^r(X^t) \Rightarrow \mathbf{s}_j(X)]$, and is identified by a `SKHandle` object that specifies (r, t, i) and another integer that specifies the target key-ID j . The basic facility provided to find a key-switching matrix are the two equivalent methods

```
const KeySwitch& getKeySwmatrix(const SKHandle& from, long toID=0) const;
const KeySwitch& getKeySwmatrix(long fromSPower, long fromXPower, long fromID=0,
long toID=0) const;
```

These methods return either a read-only reference to the requested matrix if it exists, or otherwise a reference to a dummy `KeySwitch` object that has `toKeyID` = -1. For convenience we also provide the methods `bool haveKeySwmatrix` that only test for existence, but do not return the actual matrix. Another variant is the method

```
const KeySwitch& getAnyKeySwmatrix(const SKHandle& from) const;
```

(and its counterpart `bool haveAnyKeySwmatrix`) that look for a matrix with the given source (r, t, i) and any target. All these methods first try to find the requested matrix using the `keySwitchMap` table (which is described below), and failing that they resort to linear search through the entire list of matrices.

The keySwitchMap table. Although our library supports key-switching matrices of the general form $W[\mathbf{s}_i^r(X^t) \Rightarrow \mathbf{s}_j(X)]$, we provide more facilities for finding matrices to re-linearize after automorphism (i.e., matrices of the form $W[\mathbf{s}_i(X^{t_i}) \Rightarrow \mathbf{s}_i(X)]$) than for other types of matrices.

For every secret key \mathbf{s}_i in the current instance we consider a graph G_i over the vertex set \mathbb{Z}_m^* , where we have an edge $j \rightarrow k$ if and only if we have a key-switching matrix $W[\mathbf{s}_i(X^{jk^{-1}}) \Rightarrow \mathbf{s}_i(X)]$ (where jk^{-1} is computed modulo m). We observe that if the graph G_i has a path $t \rightsquigarrow 1$ then we can apply the automorphism $X \mapsto X^t$ with re-linearization to a canonical ciphertext relative to \mathbf{s}_i as follows: Denote the path from t to 1 in the graph by

$$t = k_1 \rightarrow k_2 \cdots k_n = 1.$$

We follow the path one step at a time, for each step j applying the automorphism $X \mapsto X^{k_j k_{j+1}^{-1}}$ and then re-linearizing the result using the matrix $W[\mathbf{s}_i(X^{k_j k_{j+1}^{-1}}) \Rightarrow \mathbf{s}_i(X)]$ from the public key.

The data member `vector< vector<long> > keySwitchMap` encodes all these graphs G_i in a way that makes it easy to find the sequence of operation needed to implement any given automorphism. For every i , `keySwitchMap[i]` is a vector of indexes that stores information about the graph G_i . Specifically, `keySwitchMap[i][t]` is an index into the vector of key-switching matrices, pointing out the first step in the shortest path $t \rightsquigarrow 1$ in G_i (if any). In other words, if 1 is reachable

from t in G_i , then `keySwitchMap[i][t]` is an index k such that `keySwitching[k] = $W[s_i(X^{ts^{-1}}) \Rightarrow s_i(X)]$` where s is one step closer to 1 in G_i than t . In particular, if we have in the public key a matrix `W[s_i(X^t) \Rightarrow s_i(X)]` then `keySwitchMap[i][t]` contains the index of that matrix. If 1 is not reachable from t in G_i , then `keySwitchMap[i][t] = -1`.

The maps in `keySwitchMap` are built using a breadth-first search on the graph G_i , by calling the method `void setKeySwitchMap(long keyID=0);` This method should be called after all the key-switching matrices are added to the public key. If more matrices are generated later, then it should be called again. Once `keySwitchMap` is initialized, it is used by the method `Ctxt::smartAutomorph` as follows: to implement $X \mapsto X^t$ on a canonical ciphertext relative to secret key s_i , we do the following:

1. while $t \neq 1$
2. set $j = \text{pubKey.keySwitchMap}[i][t]$ // matrix index
3. set `matrix = pubKey.keySwitch[j]` // the matrix itself
4. set $k = \text{matrix.fromKey.getPowerOfX}()$ // the next step
5. perform automorphism $X \mapsto X^k$, then re-linearize
6. $t = t \cdot k^{-1} \bmod m$ // Now we are one step closer to 1

The operations in steps 2,3 above are combined in the method

```
const KeySwitch& FHEPubKey::getNextKSWmatrix(long t, long i);
```

That is, on input t, i it returns the matrix whose index in the list is $j = \text{keySwitchMap}[i][t]$. Also, the convenience method `bool FHEPubKey::isReachable(long t, long keyID=0) const` check if `keySwitchMap[keyID][t]` is defined, or it is -1 (meaning that 1 is not reachable from t in the graph G_{keyID}).

3.2.3 The FHESecKey class

The `FHESecKey` class is derived from `FHEPubKey`, and contains an additional data member with the secret key(s), `vector<DoubleCRT> sKeys`. It also provides methods for key-generation, decryption, and generation of key-switching matrices, as described next.

Key-generation. The `FHESecKey` class provides methods for either generating a new secret-key polynomial with a specified Hamming weight, or importing a new secret key that was generated by the calling application. That is, we have the methods:

```
long ImportSecKey(const DoubleCRT& sKey, long hwt, long ptxtSpace=0);
long GenSecKey(long hwt, long ptxtSpace=0);
```

For both these methods, if the plaintext-space modulus is unspecified then it is taken to be the default 2^r from the context. The first of these methods takes a secret key that was generated by the application and insert it into the list of secret keys, keeping track of the Hamming weight of the key and the plaintext space modulus which is supposed to be used with this key. The second method chooses a random secret key polynomial with coefficients $-1/0/1$ where exactly `hwt` of them are non-zero, then it calls `ImportSecKey` to insert the newly generated key into the list. Both of these methods return the key-ID, i.e., index of the new secret key in the list of secret keys. Also, with every new secret-key polynomial s_i , we generate and store also a key-switching matrix `W[s_i^2(X) \Rightarrow s_i(X)]` for re-linearizing ciphertexts after multiplication.

The first time that `ImportSecKey` is called for a specific instance, it also generates a public encryption key relative to this first secret key. Namely, for the first secret key \mathfrak{s} it chooses at random a polynomial $\mathfrak{c}_1^* \in_R \mathbb{A}_{Q_{\text{ct}}}$ (where Q_{ct} is the product of all the ciphertext primes) as well as a low-norm error polynomial $\mathfrak{e}^* \in \mathbb{A}_{Q_{\text{ct}}}$ (with Gaussian coefficients), then sets $\mathfrak{c}_0^* := [\text{ptxtSpace} \cdot \mathfrak{e}^* - \mathfrak{s} \cdot \mathfrak{c}_1^*]_{Q_{\text{ct}}}$. Clearly the resulting pair $(\mathfrak{c}_0^*, \mathfrak{c}_1^*)$ satisfies $\mathfrak{m}^* \stackrel{\text{def}}{=} [\mathfrak{c}_0^* + \mathfrak{s} \cdot \mathfrak{c}_1^*]_{Q_{\text{ct}}} = \text{ptxtSpace} \cdot \mathfrak{e}^*$, and the noise estimate for this public encryption key is $\text{noiseVar}^* = \mathbb{E}[|\mathfrak{m}^*(\tau_m)|^2] = p^2 \sigma^2 \cdot \phi(m)$.

Decryption. The decryption process is rather straightforward. We go over all the ciphertext parts in the given ciphertext, multiply each part by the secret key that this part points to, and sum the result modulo the current BGV modulus. Then we reduce the result modulo the plaintext-space modulus, which gives us the plaintext. This is implemented in the method

```
void Decrypt(ZZX& plaintext, const Ctxt &ciphertext) const;
```

that returns the result in the `plaintext` argument. For debugging purposes, we also provide the method `void Decrypt(ZZX& plaintext, const Ctxt &ciphertext, ZX& f) const`, that returns also the polynomial before reduction modulo the plaintext space modulus. We stress that *it would be insecure to use this method in a production system*, it is provided only for testing and debugging purposes.

Generating key-switching matrices. We also provide an interface for generating key-switching matrices, using the method:

```
void GenKeySMatrix(long fromSPower, long fromXPower, long fromKeyIdx=0,
                  long toKeyIdx=0, long ptxtSpace=0);
```

This method checks if the relevant key-switching matrix already exists, and if not then it generates it (as described in Section 3.2.1) and inserts into the list `keySwitching`. If left unspecified, the plaintext space defaults to 2^r , as defined by `context.mod2r`.

Secret-key encryption. We also provide a secret-key encryption method, that produces ciphertexts with a slightly smaller noise than the public-key encryption method. Namely we have the method

```
long FHESecKey::Encrypt(Ctxt &c, const ZX& ptxt, long ptxtSpace, long skIdx) const;
```

that encrypts the polynomial `ptxt` relative to plaintext-space modulus `ptxtSpace`, and the secret key whose index is `skIdx`. Similarly to the choice of the public encryption key, the `Encrypt` method chooses at random a polynomial $\mathfrak{c}_1 \in_R \mathbb{A}_{Q_{\text{ct}}}$ (where Q_{ct} is the product of all the ciphertext primes) as well as a low-norm error polynomial $\mathfrak{e} \in \mathbb{A}_{Q_{\text{ct}}}$ (with Gaussian coefficients), then sets $\mathfrak{c}_0 := [\text{ptxtSpace} \cdot \mathfrak{e} + \text{ptxt} - \mathfrak{s} \cdot \mathfrak{c}_1]_{Q_{\text{ct}}}$. Clearly the resulting pair $(\mathfrak{c}_0, \mathfrak{c}_1)$ satisfies $\mathfrak{m} \stackrel{\text{def}}{=} [\mathfrak{c}_0 + \mathfrak{s} \cdot \mathfrak{c}_1]_{Q_{\text{ct}}} = \text{ptxtSpace} \cdot \mathfrak{e} + \text{ptxt}$, and the noise estimate for this public encryption key is $\text{noiseVar} \approx \mathbb{E}[|\mathfrak{m}(\tau_m)|^2] = p^2 \sigma^2 \cdot \phi(m)$.

3.3 The KeySwitching module: What matrices to generate

This module implements a few useful strategies for deciding what key-switching matrices for automorphism to choose during key-generation. Specifically we have the following methods:

```
void addAllMatrices(FHESecKey& sKey, long keyID=0);
```

For $i = \text{keyID}$, generate key-switching matrices $W[\mathfrak{s}_i(X^t) \Rightarrow \mathfrak{s}_i(X)]$ for all $t \in \mathbb{Z}_m^*$.

```
void add1DMatrices(FHESecKey& sKey, long keyID=0);
```

For $i = \text{keyID}$, generate key-switching matrices $W[\mathfrak{s}_i(X^{g^e}) \Rightarrow \mathfrak{s}_i(X)]$ for every generator g of $\mathbb{Z}_m^* / \langle 2 \rangle$ with order $\text{ord}(g)$, and every exponent $0 < e < \text{ord}(g)$. Also if the order of g in \mathbb{Z}_m^* is not the same as its order in $\mathbb{Z}_m^* / \langle 2 \rangle$, then generate also the matrices $W[\mathfrak{s}_i(X^{g^{-e}}) \Rightarrow \mathfrak{s}_i(X)]$ (cf. Section 2.4).

We note that these matrices are enough to implement all the automorphisms that are needed for the data-movement routines from Section 4.

```
void addSome1DMatrices(FHESecKey& sKey, long bound=100, long keyID=0);
```

For $i = \text{keyID}$, we generate just a subset of the matrices that are generated by `add1DMatrices`, so that each of the automorphisms $X \mapsto X^{g^e}$ can be implemented by at most two steps (and similarly for $X \mapsto X^{g^{-e}}$ for generators whose orders in \mathbb{Z}_m^* and $\mathbb{Z}_m^* / \langle 2 \rangle$ are different). In other words, we ensure that the graph G_i (cf. Section 3.2.2) has a path of length at most 2 from g^e to 1 (and also from g^{-e} to 1 for g 's of different orders).

In more details, if $\text{ord}(g) \leq \text{bound}$ then we generate all the matrices $W[\mathfrak{s}_i(X^{g^e}) \Rightarrow \mathfrak{s}_i(X)]$ (or $W[\mathfrak{s}_i(X^{g^{-e}}) \Rightarrow \mathfrak{s}_i(X)]$) just like in `add1DMatrices`. When $\text{ord}(g) > \text{bound}$, however, we generate only $O(\sqrt{\text{ord}(g)})$ matrices for this generator: Denoting $B_g = \lceil \sqrt{\text{ord}(g)} \rceil$, for every $0 < e < B_g$ let $e' = e \cdot B_g \bmod m$, then we generate the matrices $W[\mathfrak{s}_i(X^{g^e}) \Rightarrow \mathfrak{s}_i(X)]$ and $W[\mathfrak{s}_i(X^{g^{e'}}) \Rightarrow \mathfrak{s}_i(X)]$. In addition, if g has a different order in \mathbb{Z}_m^* and $\mathbb{Z}_m^* / \langle 2 \rangle$ then we generate also $W[\mathfrak{s}_i(X^{g^{-e'}}) \Rightarrow \mathfrak{s}_i(X)]$.

```
void addFrbMatrices(FHESecKey& sKey, long keyID=0);
```

For $i = \text{keyID}$, generate key-switching matrices $W[\mathfrak{s}_i(X^{2^e}) \Rightarrow \mathfrak{s}_i(X)]$ for $0 < e < d$ where d is the order of 2 in \mathbb{Z}_m^* .

4 The Data-Movement Layer

At the top level of our library, we provide some interfaces that allow the application to manipulate arrays of plaintext values homomorphically. The arrays are translated to plaintext polynomials using the encoding/decoding routines provided by `PAIgebraModTwo/PAIgebraMod2r` (cf. Section 2.5), and then encrypted and manipulated homomorphically using the lower-level interfaces from the crypto layer.

4.1 The classes `EncryptedArray` and `EncryptedArrayMod2r`

These classes present the plaintext values to the application as either a linear array (with as many entries as there are elements in $\mathbb{Z}_m^* / \langle 2 \rangle$), or as a multi-dimensional array corresponding to the structure of the group $\mathbb{Z}_m^* / \langle 2 \rangle$. The difference between `EncryptedArray` and `EncryptedArrayMod2r` is that the former is used when the plaintext-space modulus is 2, while the latter is used when it is 2^r for some $r > 1$. Another difference between them is that `EncryptedArray` supports also plaintext values in binary extension fields \mathbb{F}_{2^n} , while `EncryptedArrayMod2r` only support integer plaintext values from \mathbb{Z}_{2^r} . This is reflected in the constructor for these types: For `EncryptedArray` we have the constructor


```
EncryptedArray(const FHEcontext& context, const GF2X& G=GF2X(1,1));
```

that takes as input both the context (that specifies m) and a binary polynomial G for the representation of \mathbb{F}_{2^n} (with n the degree of G). The default value for the polynomial is $G(X) = X$, resulting in plaintext values in the base field $\mathbb{F}_2 = \mathbb{Z}_2$ (i.e., individual bits). On the other hand, the constructor for `EncryptedArrayMod2r` is

```
EncryptedArrayMod2r(const FHEcontext& context);
```

that takes only the context (specifying m and the plaintext-space modulus 2^r), and the plaintext values are always in the base ring \mathbb{Z}_{2^r} . In either case, the constructors computes and store various “masks”, which are polynomials that have 1’s in some of their plaintext slots and 0 in the other slots. These masks are used in the implementation of the data movement procedures, as described below.

The multi-dimensional array view. This view arranges the plaintext slots in a multi-dimensional array, corresponding to the structure of $\mathbb{Z}_m^*/\langle 2 \rangle$. The number of dimensions is the same as the number of generators that we have for $\mathbb{Z}_m^*/\langle 2 \rangle$, and the size along the i ’th dimension is the order of the i ’th generator.

Recall from Section 2.4 that each plaintext slot is represented by some $t \in \mathbb{Z}_m^*$, such that the set of representatives $T \subset \mathbb{Z}_m^*$ has exactly one element from each conjugacy class of $\mathbb{Z}_m^*/\langle 2 \rangle$. Moreover, if $f_1, \dots, f_n \in T$ are the generators of $\mathbb{Z}_m^*/\langle 2 \rangle$ (with f_i having order $\text{ord}(f_i)$), then every $t \in T$ can be written uniquely as $t = [\prod_i f_i^{e_i}]_m$ with each exponent e_i taken from the range $0 \leq e_i < \text{ord}(f_i)$. The generators are roughly arranged by their order (i.e., $\text{ord}(f_i) \geq \text{ord}(f_{i+1})$), except that we put all the generators that have the same order in \mathbb{Z}_m^* and $\mathbb{Z}_m^*/\langle 2 \rangle$ before all the generators that have different orders in the two groups.

Hence the multi-dimensional-array view of the plaintext slots will have them arranged in a n -dimensional hypercube, with the size of the i ’th side being $\text{ord}(f_i)$. Every entry in this hypercube is indexed by some $\vec{e} = (e_1, e_2, \dots, e_n)$, and it contains the plaintext slot associated with the representative $t = [\prod_i f_i^{e_i}]_m \in T$. (Note that the lexicographic order on the vectors \vec{e} of indexes induces a linear ordering on the plaintext slots, which is what we use in our linear-array view described below.) The multi-dimensional-array view provides the following interfaces:

```
long dimension() const; returns the dimensionality (i.e., the number of generators in  $\mathbb{Z}_m^*/\langle 2 \rangle$ ).
```

```
long sizeOfDimension(long i); returns the size along the  $i$ ’th dimension (i.e.,  $\text{ord}(f_i)$ ).
```

```
long coordinate(long i, long k) const; return the  $i$ ’th entry of the  $k$ ’th vector in lexicographic order.
```

```
void rotate1D(Ctxt& ctxt, long i, long k) const;
```

This method rotates the hypercube by k positions along the i ’th dimension, moving the content of the slot indexed by $(e_1 \dots, e_i, \dots, e_n)$ to the slot indexed by $(e_1 \dots, e_i + k, \dots, e_n)$, addition modulo $\text{ord}(f_i)$. Note that the argument k above can be either positive or negative, and rotating by $-k$ is the same as rotating by $\text{ord}(f_i) - k$.

The `rotate` operation is closely related to the “native” automorphism operation of the lower-level `Ctxt` class. Indeed, if f_i has the same order in \mathbb{Z}_m^* as in $\mathbb{Z}_m^*/\langle 2 \rangle$ then we just apply the automorphism $X \mapsto X^{f_i^k}$ on the input ciphertext using `ctxt.smartAutomorph(f_i^k)`. If f_i has different orders in \mathbb{Z}_m^* and $\mathbb{Z}_m^*/\langle 2 \rangle$ then we need to apply the two automorphisms $X \mapsto X^{f_i^k}$

and $X \mapsto X^{f_i^{k-\text{ord}(f_i)}}$ and then “mix and match” the two resulting ciphertexts to pick from each of them the plaintext slots that did not undergo wraparound (see description of the `select` method below).

```
void shift1D(Ctxt& ctxt, long i, long k) const;
```

This is similar to `rotate1D`, except it implements a non-cyclic shift with zero fill. Namely, for a positive $k > 0$, the content of any slot indexed by $(e_1 \dots, e_i, \dots, e_n)$ with $e_i < \text{ord}(f_i) - k$ is moved to the slot indexed by $(e_1 \dots, e_i + k, \dots, e_n)$, and all the other slots are filled with zeros. For a negative $k < 0$, the content of any slot indexed by $(e_1 \dots, e_i, \dots, e_n)$ with $e_i \geq |k|$ is moved to the slot indexed by $(e_1 \dots, e_i + k, \dots, e_n)$, and all the other slots are filled with zeros.

The operation is implemented by applying the corresponding automorphism(s), and then zero-ing out the wraparound slots by multiplying the result by a constant polynomial that has zero in these slots.

The linear array view. This view arranges the plaintext slots in a linear array, with as many entries as there are plaintext slots (i.e., $|\mathbb{Z}_m^* / \langle 2 \rangle|$). These entries are ordered according to the lexicographic order on the vectors of indexes from the multi-dimensional array view above. In other words, we obtain a linear array simply by “opening up” the hypercube from above in lexicographic order. The linear-array view provides the following interfaces:

```
long size() const; returns the number of entries in the array, i.e., the number of plaintext slots.
```

```
void rotate(Ctxt& ctxt, long k) const;
```

Cyclically rotate the linear array by k positions, moving the content of the j ’th slot (by the lexicographic order) to slot $j + k$, addition modulo the number of slots. (Below we denote the number of slots by N .) Rotation by a negative number $-N < k < 0$ is the same as rotation by the positive amount $k + N$.

The procedure for implementing this cyclic rotation is roughly a concurrent version of the grade-school addition-with-carry procedure, building on the multidimensional rotations from above. What we need to do is to add k (modulo N) to the index of each plaintext slot, all in parallel. To that end, we think of the indexes (and the rotation amount k) as they are represented in the lexicographic order above. Namely, we identify k with the vector $\vec{e}^{(k)} = (e_1^{(k)}, \dots, e_n^{(k)})$ which is k ’th in the lexicographic order (and similarly identify each index j with the j ’th vector in that order). We can now think of rotation by k as adding the multi-precision vector $\vec{e}^{(k)}$ to all the vectors $\vec{e}^{(j)}$, $j = 0, 1, \dots, N - 1$ in parallel.

Beginning with the least-significant digit in these vector, we use rotate-by- $e_n^{(k)}$ along the n ’th dimension to implement the operation of $e_n^{(j)} = e_n^{(j)} + e_n^{(k)} \bmod \text{ord}(f_n)$ for all j at once.

Moving to the next digit, we now have to add to each $e_{n-1}^{(j)}$ either $e_{n-1}^{(k)}$ or $1 + e_{n-1}^{(k)}$, depending on whether or not there was a carry from the previous position. To do that, we compute two rotation amount along the $(n - 1)$ ’th dimension, by $e_{n-1}^{(k)}$ and $1 + e_{n-1}^{(k)}$, then use a MUX operation to choose the right rotation amount for every slot. Namely, indexes j for which $e_n^{(j)} \geq \text{ord}(f_i) - e_n^{(k)}$ (so we have a carry) are taken from the copy that was rotated by $1 + e_{n-1}^{(k)}$, while other indexes j are taken from the copy that was rotated by $e_{n-1}^{(k)}$.

The MUX operation is implemented by preparing a constant polynomial that has 1's in the slots corresponding to indexes (e_1, \dots, e_n) with $e_n \geq \text{ord}(f_i) - e_n^{(k)}$ and 0's in all the other slots (call this polynomial **mask**), then computing $\vec{c}' = \vec{c}_1 \cdot \text{mask} + \vec{c}_2 \cdot (1 - \text{mask})$, where \vec{c}_1, \vec{c}_2 are the two ciphertexts generated by rotation along dimension $n - 1$ by $1 + e_{n-1}^{(k)}$ and $e_{n-1}^{(k)}$, respectively.

We then move to the next digit, preparing a mask for those j 's for which we have a carry into that position, then rotating by $1 + e_{n-2}^{(k)}$ and $e_{n-2}^{(k)}$ along the $(n - 2)$ 'nd dimension and using the mask to do the MUX between these two ciphertexts. We proceed in a similar manner until the most significant digit. To complete the description of the algorithm, note that the mask for processing the i 'th digit is computed as follows: For each index j , which is represented by the vector $(e_1^{(j)}, \dots, e_i^{(j)}, \dots, e_n^{(j)})$, we have $\text{mask}_i[j] = 1$ if either $e_i^{(j)} \geq \text{ord}(f_i) - e_i^{(k)}$, or if $e_i^{(j)} = \text{ord}(f_i) - e_i^{(k)} - 1$ and $\text{mask}_{i-1}[j] = 1$ (i.e. we had a carry from position $i - 1$ to position i). Hence the rotation procedure works as follows:

Rotate(\vec{c}, k):

0. Let $(e_1^{(k)}, \dots, e_n^{(k)})$ be the k 'th vector in lexicographic order.
1. $M_n := \text{all-1 mask}$ // M_n is a polynomial with 1 in all the slots
2. Rotate \vec{c} by $e_n^{(k)}$ along the n 'th dimension
3. For $i = n - 1$ down to 1
4. $M'_i := 1$ in the slots j with $e_{i+1}^{(j)} \geq \text{ord}(f_{i+1}) - e_{i+1}^{(k)}$, 0 in all the other slots
5. $M''_i := 1$ in the slots j with $e_{i+1}^{(j)} = \text{ord}(f_{i+1}) - e_{i+1}^{(k)} - 1$, 0 in all the outer slots
6. $M_i := M'_i + M''_i \cdot M_{i+1}$ // The i 'th mask
7. $\vec{c}' := \text{rotate } \vec{c} \text{ by } e_i^{(k)} \text{ along the } i\text{'th dimension}$
8. $\vec{c}'' := \text{rotate } \vec{c} \text{ by } 1 + e_i^{(k)} \text{ along the } i\text{'th dimension}$
9. $\vec{c} := \vec{c}'' \cdot M_i + \vec{c}' \cdot (1 - M_i)$
10. Return \vec{c} .

void shift(Ctxt& ctxt, long k) const; Non-cyclic shift of the linear array by k positions, with zero-fill. For a positive $k > 0$, then every slot $j \geq k$ gets the content of slot $j - k$, and every slot $j < k$ gets zero. For a negative $k < 0$, every slot $j < N - |k|$ gets the content of slot $j + |k|$, and every slot $j \geq N - |k|$ gets zero (with N the number of slots).

For $k > 0$, this procedure is implemented very similarly to the **rotate** procedure above, except that in the last iteration (processing the most-significant digit) we replace the operation of rotate-by- $e_1^{(k)}$ along the 1'st dimension by shift-by- $e_1^{(k)}$ along the 1'st dimension (and similarly use shift-by- $(1 + e_1^{(k)})$ rather than rotate-by- $(1 + e_1^{(k)})$). For a negative amount $-N < k < 0$, we use the same procedure upto the last iteration with amount $N + k$, and in the last iteration use shift-by- e' and shift-by- $(1 + e')$ along the 1st dimension, for the negative number $e' = e_1^{(k)} - \text{ord}(f_i)$.

Other operations. In addition to the following rotation methods, the classes **EncryptedArray** and **EncryptedArrayMod2r** also provide convenience methods that handle both encoding and homomorphic operations in one shot. The class **EncryptedArray** uses type **vector<GF2X>** for a plaintext array

(since the plaintext values can be elements in an extension field \mathbb{F}_{2^n}), whereas class `EncryptedArrayMod2r` uses type `vector<long>` for the same purpose (since the plaintext values in this case are integers). The methods that are provided are the following:

```
// Fill the array with random plaintext data
void random(vector<GF2X>& array) const;      // EncryptedArray
void random(vector<long>& array) const;      // EncryptedArrayMod2r

// Encode the given array in a polynomial
void encode(ZZX& ptxt, const vector<GF2X>& array) const;
void encode(ZZX& ptxt, const vector<long>& array) const;

// Decode the given polynomial into an array of plaintext values
void decode(vector<long>& array, const ZX& ptxt) const
void decode(vector<GF2X>& array, const ZX& ptxt) const

// Multiply by the ciphertext by a polynomial encoding the given array
void multByConst(Ctxt& ctxt, const vector<GF2X>& array);
void multByConst(Ctxt& ctxt, const vector<long>& array);

// Add to the ciphertext a polynomial encoding the given array
void addConst(Ctxt& ctxt, const vector<GF2X>& array)
void addConst(Ctxt& ctxt, const vector<long>& array)

// MUX: for p=encode(selector), set c1 = c1*p + c2*(1-p)
void select(Ctxt& c1, const Ctxt& c2, const vector<GF2X>& selector) const;
void select(Ctxt& c1, const Ctxt& c2, const vector<long>& selector) const;

// Encode the array in a polynomial, then encrypt it in the ciphertext c
void encrypt(Ctxt& c, const FHEPubKey& pKey, const vector<GF2X>& array) const;
void encrypt(Ctxt& c, const FHEPubKey& pKey, const vector<long>& array) const;

// Decrypt the ciphertext c, then decode the result into the array
void decrypt(const Ctxt& c, const FHESecKey& sKey, vector<GF2X>& array) const;
void decrypt(const Ctxt& c, const FHESecKey& sKey, vector<long>& array) const;
```

5 Using the Library

Below we provide two examples of how this library can be used by an application program. These examples compute a simple circuit with homomorphic arithmetic over either $GF(2^8)$ (represented using the AES polynomial, $G(X) = X^8 + X^4 + X^3 + X + 1$), or over \mathbb{Z}_{2^5} .

5.1 Homomorphic Operations over $GF(2^8)$

```
/** Determine the parameters (cf. [5, Appendix C]) */

long ptxtSpace = 2;
long nDigits = 2; // # of digits/# of columns in key-switching matrices
long k = 80;      // security parameter
long weight = 64; // Hamming weight of secret keys
long lvls = 3;    // number of ciphertext-primes in the modulus chain
long m = 11441;   // the parameter m, defining  $Z_m^*$  and  $\Phi_m(X)$ 

/** Setup the various tables, and choose the keys */

FHEcontext context(m); // initialize a new context for the parameter m
buildModChain(context, lvls, nDigits); // build the modulus chain

FHESecKey secretKey(context); // initialize a secret-key object
const FHEPubKey& publicKey = secretKey; // use the same object as a public-key
secretKey.GenSecKey(weight, ptxtSpace); // draw a random secret key

addSome1DMatrices(secretKey); // compute some key-switching matrices
// We could also use add1DMatrices instead of addSome1DMatrices

GF2X G; // G is the AES polynomial,  $G(X) = X^8 + X^4 + X^3 + X + 1$ 
SetCoeff(G, 8); SetCoeff(G, 4); SetCoeff(G, 3); SetCoeff(G, 1); SetCoeff(G, 0);

EncryptedArray ea(context, G); // An EncryptedArray object, encoding wrt G
long nslots = ea.size(); // number of plaintext slots

/** Encrypt random arrays over  $GF(2^8)$  */

vector<GF2X> p0, p1, p2, p3; // Choose random arrays
ea.random(p0);
ea.random(p1);
ea.random(p2);
ea.random(p3);

vector<GF2X> const1, const2; // two more random "constants"
ea.random(const1);
ea.random(const2);

ZZX const1_poly, const2_poly; // encode constants as polynomials
ea.encode(const1_poly, const1);
ea.encode(const2_poly, const2);

// Encrypt the random arrays
Ctxt c0(publicKey), c1(publicKey), c2(publicKey), c3(publicKey);
ea.encrypt(c0, publicKey, p0);
ea.encrypt(c1, publicKey, p1);
ea.encrypt(c2, publicKey, p2);
ea.encrypt(c3, publicKey, p3);
```

```

/** Perform homomorphic operations */

c1.multiplyBy(c0);           // also does mod-switching, key-switching
c0.addConstant(const1_poly);
c2.multByConstant(const2_poly);

Ctxt tmp(c1);               // tmp = c1
long amt = RandomBnd(2*(nslots/2)+1)-(nslots/2); // in [-nslots/2..nslots/2]
ea.shift(tmp, amt);         // rotate tmp by amt
c2 += tmp;                  // then add to c2

amt = RandomBnd(2*nslots-1) - (nslots-1);          // in [-(nslots-1)..nslots-1]
ea.rotate(c2, amt);

c1.negate();
c3.multiplyBy(c2);
c0 -= c3;

/** Decrypt the results of the computation */

ea.decrypt(c0, secretKey, pp0);
ea.decrypt(c1, secretKey, pp1);
ea.decrypt(c2, secretKey, pp2);
ea.decrypt(c3, secretKey, pp3);

```

5.2 Homomorphic Operations over \mathbb{Z}_{2^5}

This example is almost identical to the previous one, except that the `FHEcontext` is initialized also with the parameter $r = 5$, we use `EncryptedArrayMod2r` instead of `EncryptedArray` and `vector<long>` instead of `vector<GF2X>`, and we do not need the polynomial G .

```

/** Determine the parameters (cf. [5, Appendix C]) */

long r = 5;
long ptxtSpace = 1L << r; // plaintext space modulo 2^5
long nDigits = 2; // # of digits/# of columns in key-switching matrices
long k = 80;      // security parameter
long weight = 64; // Hamming weight of secret keys
long lvls = 7;    // number of ciphertext-primes in the modulus chain
long m = 11441;   // the parameter m, defining Zm^* and Phi_m(X)

/** Setup the various tables, and choose the keys */

FHEcontext context(m, r); // initialize a new context for the parameters m,r
buildModChain(context, lvls, nDigits); // build the modulus chain
FHESecKey secretKey(context); // initialize a secret-key object
const FHEPubKey& publicKey = secretKey; // use the same object as a public-key
secretKey.GenSecKey(weight, ptxtSpace); // draw a random secret key

addSome1DMatrices(secretKey); // compute some key-switching matrices
// We could also use add1DMatrices instead of addSome1DMatrices

```

```

EncryptedArrayMod2r ea(context);          // An EncryptedArrayMod2r object
long nslots = ea.size();                  // number of plaintext slots

/**/ Encrypt random arrays over  $\mathbb{Z}_{32}$  /**/

vector<long> p0, p1, p2, p3; // Choose random arrays
ea.random(p0);
ea.random(p1);
ea.random(p2);
ea.random(p3);

vector<long> const1, const2; // two more random "constants"
ea.random(const1);
ea.random(const2);

ZZX const1_poly, const2_poly; // encode constants as polynomials
ea.encode(const1_poly, const1);
ea.encode(const2_poly, const2);

// Encrypt the random arrays
Ctxt c0(publicKey), c1(publicKey), c2(publicKey), c3(publicKey);
ea.encrypt(c0, publicKey, p0);
ea.encrypt(c1, publicKey, p1);
ea.encrypt(c2, publicKey, p2);
ea.encrypt(c3, publicKey, p3);

/**/ Perform homomorphic operations /**/

c1.multiplyBy(c0);          // also does mod-switching, key-switching
c0.addConstant(const1_poly);
c2.multByConstant(const2_poly);

Ctxt tmp(c1);          // tmp = c1
long amt = RandomBnd(2*(nslots/2)+1)-(nslots/2); // in  $[-nslots/2..nslots/2]$ 
ea.shift(tmp, amt); // rotate tmp by amt
c2 += tmp;          // then add to c2

amt = RandomBnd(2*nslots-1) - (nslots-1);          // in  $[-(nslots-1)..nslots-1]$ 
ea.rotate(c2, amt);

c1.negate();
c3.multiplyBy(c2);
c0 -= c3;

/**/ Decrypt the results of the computation /**/

vector<long> pp0, pp1, pp2, pp3;
ea.decrypt(c0, secretKey, pp0);
ea.decrypt(c1, secretKey, pp1);
ea.decrypt(c2, secretKey, pp2);
ea.decrypt(c3, secretKey, pp3);

```

References

- [1] L. I. Bluestein. A linear filtering approach to the computation of the discrete fourier transform. Northeast Electronics Research and Engineering Meeting Record 10, 1968.
- [2] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. Fully homomorphic encryption without bootstrapping. In *Innovations in Theoretical Computer Science (ITCS'12)*, 2012. Available at <http://eprint.iacr.org/2011/277>.
- [3] C. Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st ACM Symposium on Theory of Computing – STOC 2009*, pages 169–178. ACM, 2009.
- [4] C. Gentry, S. Halevi, and N. Smart. Fully homomorphic encryption with polylog overhead. In *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 465–482. Springer, 2012. Full version at <http://eprint.iacr.org/2011/566>.
- [5] C. Gentry, S. Halevi, and N. Smart. Homomorphic evaluation of the AES circuit. In *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 850–867. Springer, 2012. Full version at <http://eprint.iacr.org/2012/099>.
- [6] C. Gentry, S. Halevi, and N. P. Smart. Better bootstrapping in fully homomorphic encryption. In *Public Key Cryptography – PKC 2012*, volume 7293 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2012.
- [7] V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. In H. Gilbert, editor, *Advances in Cryptology – EUROCRYPT'10*, volume 6110 of *Lecture Notes in Computer Science*, pages 1–23. Springer, 2010.
- [8] R. Rivest, L. Adleman, and M. Dertouzos. On data banks and privacy homomorphisms. In *Foundations of Secure Computation*, pages 169–177. Academic Press, 1978.
- [9] V. Shoup. NTL: A Library for doing Number Theory. <http://shoup.net/ntl/>, Version 5.5.2, 2010.
- [10] N. P. Smart and F. Vercauteren. Fully homomorphic SIMD operations. Manuscript at <http://eprint.iacr.org/2011/133>, 2011.

A Proof of noise-estimate

Recall that we observed empirically that for a random Hamming-weight- H polynomial \mathfrak{s} with coefficients $-1/0/1$ and an integral power r we have $\mathbb{E}[|\mathfrak{s}^r(\tau)|^{2r}] \approx r! \cdot H^r$, where τ is the principal complex m -th root of unity, $\tau = e^{2\pi i/m}$.

To simplify the proof, we analyze the case that each coefficient of \mathfrak{s} is chosen uniformly at random from $-1/0/1$, so that the expected Hamming weight is H . Also, we assume that \mathfrak{s} is chosen as a degree- $(m-1)$ polynomial (rather than degree $\phi(m)-1$).

Theorem 1. *Suppose m, r, H are positive integers, with $H \leq m$, and let $\tau = e^{2\pi i/m} \in \mathbb{C}$. Suppose that we choose f_0, \dots, f_{m-1} independently, where for $i = 0, \dots, m-1$, f_i is ± 1 with probability*

$H/2m$ each and 0 with probability $1 - H/m$. Let $f(X) = \sum_{i=0}^{m-1} f_i X^i$. Then for fixed r and $H, m \rightarrow \infty$, we have

$$\mathbb{E}[|f(\tau)|^{2r}] \sim r! H^r.$$

In particular, for $H \geq 2r^2$, we have

$$\left| \frac{\mathbb{E}[|f(\tau)|^{2r}]}{r! H^r} - 1 \right| \leq \frac{2r^2}{H} + \frac{2^{r+1} r^2}{m}.$$

Before proving Theorem 1, we introduce some notation and prove some technical results that will be useful.

Recall the “falling factorial” notation: for integers n, k with $0 \leq k \leq n$, we define $n^{\underline{k}} = \prod_{j=0}^{k-1} (n - j)$.

Lemma 1. For $n \geq k^2 > 0$, we have $n^k - n^{\underline{k}} \leq k^2 n^{k-1}$.

Proof. We have

$$n^{\underline{k}} \geq (n - k)^k = n^k - \binom{k}{1} k n^{k-1} + \binom{k}{2} k^2 n^{k-2} - \binom{k}{3} k^3 n^{k-3} + \dots.$$

The lemma follows by verifying that when $n \geq k^2$, in the above binomial expansion, the sum of every consecutive positive/negative pair of terms is non-negative. \square

Lemma 2. For $n \geq 2k^2 > 0$, we have $n^k \leq 2n^{\underline{k}}$.

Proof. This follows immediately from the previous lemma. \square

Next, we recall the notion of the *Stirling number of the second kind*, which is the number of ways to partition a set of ℓ objects into k non-empty subsets, and is denoted $\left\{ \begin{smallmatrix} \ell \\ k \end{smallmatrix} \right\}$. We use the following standard result:

$$\sum_{k=1}^{\ell} \left\{ \begin{smallmatrix} \ell \\ k \end{smallmatrix} \right\} n^{\underline{k}} = n^{\ell}. \quad (1)$$

Finally, we define M_{2n} to be the number of perfect matchings in the complete graph on $2n$ vertices, and $M_{n,n}$ to be the number of perfect matchings on the complete bipartite graph on two sets of n vertices. The following facts are easy to establish:

$$M_{n,n} = n! \quad (2)$$

and

$$M_{2n} \leq 2^n n!. \quad (3)$$

We now turn to the proof of the theorem. We have

$$f(\tau)^{2r} = f(\tau)^r f(\bar{\tau})^r = \sum_{i_1, \dots, i_{2r}} f_{i_1} \cdots f_{i_{2r}} \cdot \tau^{i_1} \cdots \tau^{i_r} \cdot \tau^{-i_{r+1}} \cdots \tau^{-i_{2r}}.$$

We will extend the usual notion of expected values to complex-valued random variables: if U and V are real-valued random variables, then $\mathbb{E}[U + V\mathbf{i}] = \mathbb{E}[U] + \mathbb{E}[V]\mathbf{i}$. The usual rules for sums and products of expectations work equally well. By linearity of expectation, we have

$$\mathbb{E}[f(\tau)^{2r}] = \sum_{i_1, \dots, i_{2r}} \mathbb{E}[f_{i_1} \cdots f_{i_{2r}}] \cdot \tau^{i_1} \cdots \tau^{i_r} \cdot \tau^{-i_{r+1}} \cdots \tau^{-i_{2r}}. \quad (4)$$

Here, each index i_t runs over the set $\{0, \dots, m-1\}$. In this sum, because of independence and the fact that any odd power of f_i has expected value 0, the only terms that contribute a non-zero value are those in which each index value occurs an even number of times, in which case, if there are k distinct values among i_1, \dots, i_{2r} , we have

$$\mathbb{E}[f_{i_1} \cdots f_{i_{2r}}] = (H/m)^k.$$

We want to regroup the terms in (4). To this end, we introduce some notation: for an integer $t \in \{1, \dots, 2r\}$ define $w(t) = 1$ if $t \leq r$, and $w(t) = -1$ if $t > r$; for a subset $e \subseteq \{1, \dots, 2r\}$, define $w(e) = \sum_{t \in e} w(t)$. We call $w(e)$ the “weight” of e . Then we have:

$$\mathbb{E}[f(\tau)^{2r}] = \sum_{P=\{e_1, \dots, e_k\}} (H/m)^k \sum'_{j_1, \dots, j_k} \tau^{j_1 w(e_1) + \dots + j_k w(e_k)}. \quad (5)$$

Here, the outer summation is over all “even” partitions $P = \{e_1, \dots, e_k\}$ of the set $\{1, \dots, 2r\}$, where each element of the partition has an even cardinality. The inner summation is over all sequences of indices j_1, \dots, j_k , where each index runs over the set $\{0, \dots, m-1\}$, but where no value in the sequence is repeated — the special summation notation \sum'_{j_1, \dots, j_k} emphasizes this restriction.

Since $|\tau| = 1$, it is clear that

$$\left| \mathbb{E}[f(\tau)^{2r}] - \sum_{P=\{e_1, \dots, e_k\}} (H/m)^k \sum_{j_1, \dots, j_k} \tau^{j_1 w(e_1) + \dots + j_k w(e_k)} \right| \leq \sum_{P=\{e_1, \dots, e_k\}} (H/m)^k (m^k - m^{\underline{k}}) \quad (6)$$

Note that in this inequality the inner sum on the left is over *all* sequences of indices j_1, \dots, j_k , without the restriction that the indices in the sequence are unique.

Our first task is to bound the sum on the right-hand side of (6). Observe that any even partition $P = \{e_1, \dots, e_k\}$ can be formed by merging the edges of some perfect matching on the complete graph on vertices $\{1, \dots, 2r\}$. So we have

$$\begin{aligned} \sum_{P=\{e_1, \dots, e_k\}} (H/m)^k (m^k - m^{\underline{k}}) &\leq \sum_{P=\{e_1, \dots, e_k\}} (H/m)^k k^2 m^{k-1} \quad (\text{by Lemma 1}) \\ &\leq \frac{r^2}{m} \sum_{P=\{e_1, \dots, e_k\}} H^k \\ &\leq \frac{r^2}{m} M_{2r} \sum_{k=1}^r \left\{ \begin{matrix} r \\ k \end{matrix} \right\} H^k \quad (\text{partitions formed from matchings}) \\ &\leq \frac{r^2 2^r r!}{m} \sum_{k=1}^r \left\{ \begin{matrix} r \\ k \end{matrix} \right\} H^k \quad (\text{by (3)}) \\ &\leq \frac{r^2 2^{r+1} r!}{m} \sum_{k=1}^r \left\{ \begin{matrix} r \\ k \end{matrix} \right\} H^{\underline{k}} \quad (\text{by Lemma 2}) \\ &= \frac{r^2 2^{r+1} r!}{m} H^r \quad (\text{by 1}). \end{aligned}$$

Combining this with (6), we have

$$\left| \mathbb{E}[f(\tau)^{2r}] - \sum_{P=\{e_1, \dots, e_k\}} (H/m)^k \sum_{j_1, \dots, j_k} \tau^{j_1 w(e_1) + \dots + j_k w(e_k)} \right| \leq r! H^r \cdot \frac{2^{r+1} r^2}{m}. \quad (7)$$

So now consider the inner sum in (7). The weights $w(e_1), \dots, w(e_k)$ are integers bounded by r in absolute value, and r is strictly less than m by the assumption $2r^2 \leq H \leq m$. If any weight, say $w(e_1)$, is non-zero, then $\tau^{w(e_1)}$ has multiplicative order dividing m , but not 1, and so the sum $\sum_j \tau^{jw(e_1)}$ vanishes, and hence

$$\sum_{j_1, \dots, j_k} \tau^{j_1 w(e_1) + \dots + j_k w(e_k)} = \left(\sum_{j_1} \tau^{j_1 w(e_1)} \right) \left(\sum_{j_2, \dots, j_k} \tau^{j_2 w(e_2) + \dots + j_k w(e_k)} \right) = 0.$$

Otherwise, if all the weights are $w(e_1), \dots, w(e_k)$ are zero, then

$$\sum_{j_1, \dots, j_k} \tau^{j_1 w(e_1) + \dots + j_k w(e_k)} = m^k.$$

We therefore have

$$\sum_{P=\{e_1, \dots, e_k\}} (H/m)^k \sum_{j_1, \dots, j_k} \tau^{j_1 w(e_1) + \dots + j_k w(e_k)} = \sum_{\substack{P=\{e_1, \dots, e_k\} \\ w(e_1)=\dots=w(e_k)=0}} H^k, \quad (8)$$

Observe that any partition $P = \{e_1, \dots, e_k\}$ with $w(e_1) = \dots = w(e_k) = 0$ can be formed by merging the edges of some perfect matching on the complete bipartite graph with vertex sets $\{1, \dots, r\}$ and $\{r+1, \dots, 2r\}$. The total number of such matchings is $r!$ (see (2)). So we have

$$\begin{aligned} r!H^r &\leq \sum_{\substack{P=\{e_1, \dots, e_k\} \\ w(e_1)=\dots=w(e_k)=0}} H^k \leq r!H^r + r! \sum_{k=1}^{r-1} \left\{ \begin{matrix} r \\ k \end{matrix} \right\} H^k \\ &\leq 2r! \sum_{k=1}^{r-1} \left\{ \begin{matrix} r \\ k \end{matrix} \right\} H^k \quad (\text{by Lemma 2}) \\ &= 2r!(H^r - H^r) \quad (\text{by (1)}) \\ &\leq 2r!r^2 H^{r-1} \quad (\text{by Lemma 1}) \end{aligned}$$

Combining this with (7) and (8) proves the theorem.

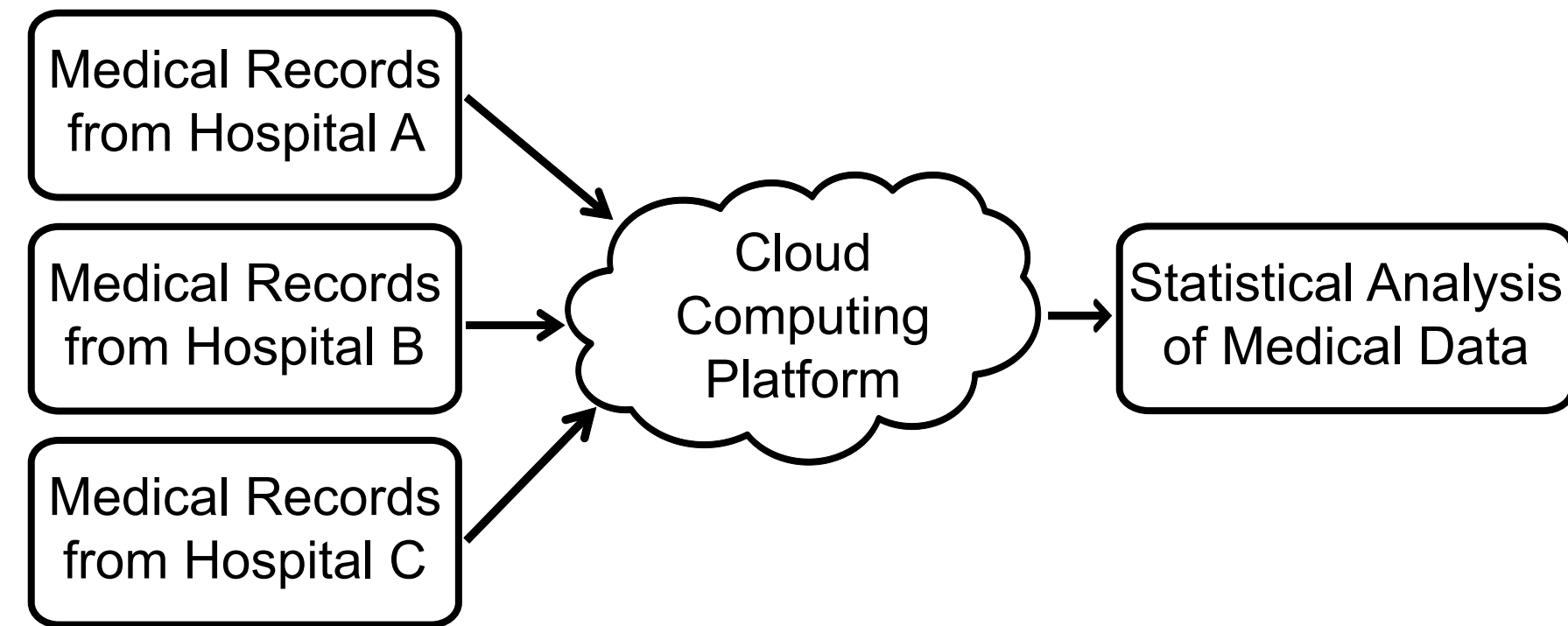


Using Homomorphic Encryption for Large Scale Statistical Analysis

CURIS 2012
David Wu and Jacob Haven
Advised by Professor Dan Boneh

Motivation

Cloud-based solutions have become increasingly popular in the past few years. An example of the cloud-based model is shown below. Here, three different hospitals provide data to the cloud. The cloud computing platform then analyzes and extracts useful information from the data.



One of the main concern with cloud computing has been the privacy and confidentiality of the data. One solution is to send the data encrypted to the cloud. However, we still need to support useful computations on the encrypted data. Fully homomorphic encryption (FHE) is a way of supporting such computations on encrypted data.

We note that while other mechanisms exist for secure computation, they generally require the different data providers to exchange information. Because FHE schemes are *public key* schemes, FHE is much better suited for the scenario where we have many sources of data.

Our Approach

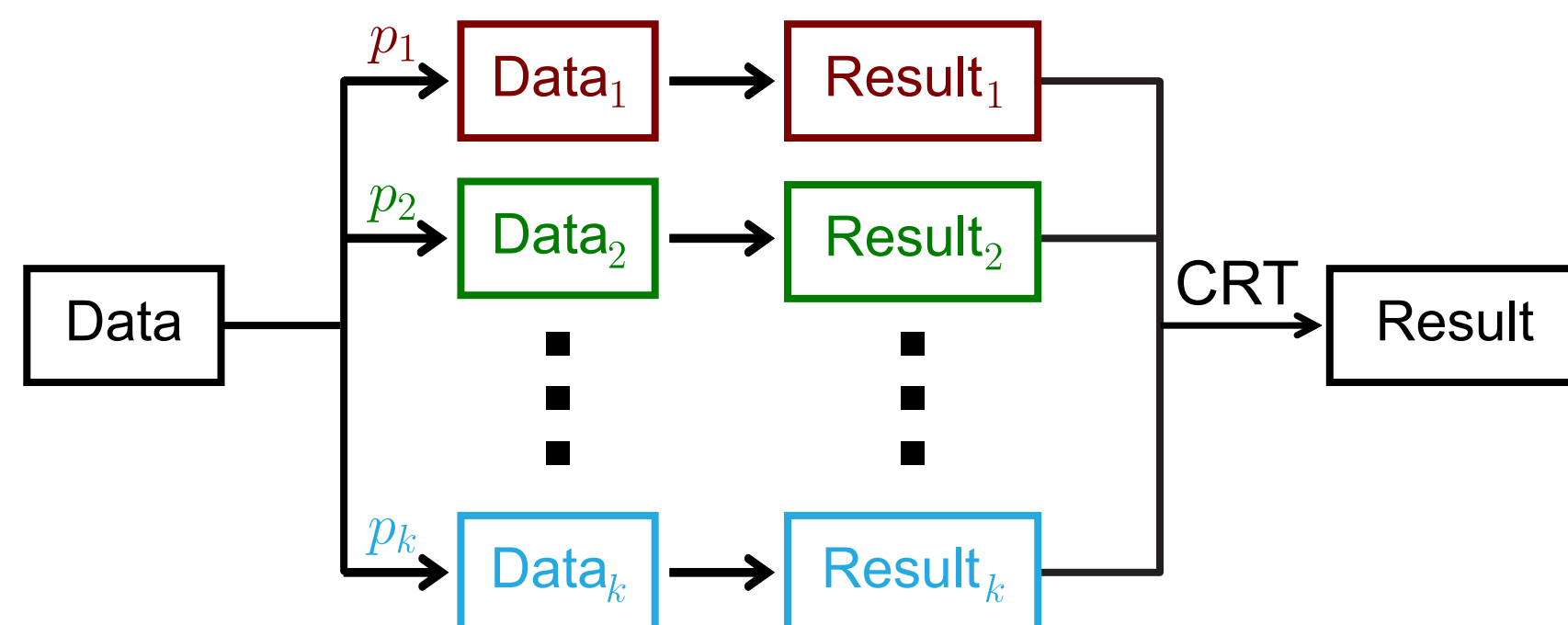
Due to the significant overhead in homomorphic computation, implementations of homomorphic encryption schemes for statistical analysis have been limited to small datasets (≈ 100 data points) and low dimensional data (≈ 2 -4 dimensions).

Using recent techniques in batched computation and a different message encoding scheme, we demonstrate the viability of using leveled homomorphic encryption to compute on datasets with over a million elements as well as datasets of much higher dimension.

In particular, we consider two applications of homomorphic encryption: computing the mean and covariance of multivariate data and performing linear regression over encrypted datasets.

Computation over Large Integers

To support computation over large amounts of data, we need to be able to handle large integers (i.e., 128-bit precision). However, it is not computationally feasible to choose message spaces of this magnitude. To support computations with at least 128-bit precision, we leverage the Chinese Remainder Theorem (CRT):

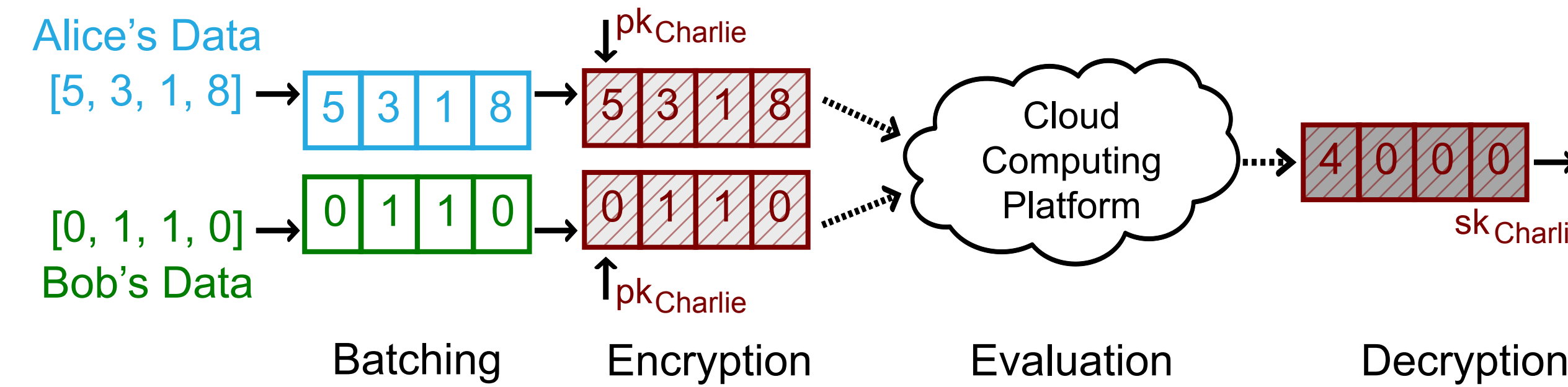


- We choose primes p_1, p_2, \dots, p_k such that $p_1 p_2 \dots p_k > 2^{128}$.
- We perform the computation modulo each prime. Given the results of the computation with respect to each prime, we apply the CRT to obtain the value modulo the product of the primes (at least 128-bit precision).
- The computations with respect to each prime is completely independent of the computation with respect to the other primes. As such, all of the computations are naturally parallelizable.

Homomorphic Encryption Scheme (Client Side)

Leveled fully homomorphic encryption (FHE) schemes supports addition and multiplication over ciphertexts. Such schemes are capable of evaluating boolean circuits with bounded depth (determined by the number of multiplications) over ciphertexts, and thus, can perform many computations over the ciphertext.

Consider a scenario where Charlie wants to compute the inner product of Alice's and Bob's data. Note that covariance computation and linear regression can be expressed in terms of matrix products, which can be viewed as a series of inner products.

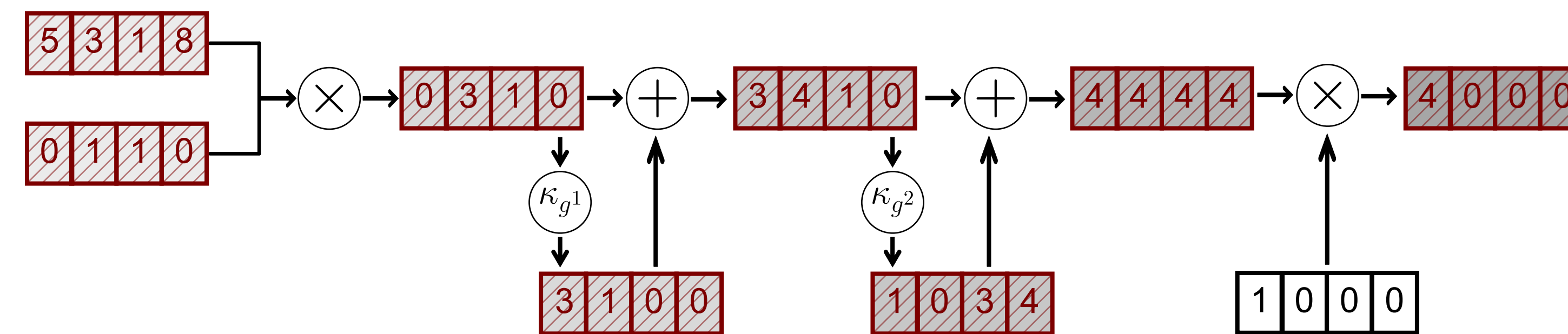


Clear blocks represent plaintext blocks while striped blocks represent ciphertext blocks. The shading in the ciphertext block represents the amount of noise in the ciphertext (explained below).

- **Batching:** Pack multiple plaintext messages (data elements) into a single ciphertext block. This enables the server to evaluate a single instruction on multiple data with low overhead.
- **Encryption:** Encrypt the packed plaintext blocks with the FHE public key (Charlie's public key).
- **Evaluation:** Send the encrypted data to the cloud server for processing.
- **Decryption:** The client (Charlie) decrypts the result using his secret key.

Homomorphic Encryption Scheme (Server Side)

Our leveled FHE scheme supports three basic operations: addition, multiplication, and Frobenius automorphisms. Below, we show how we can use these operations to compute the inner product on encrypted data.



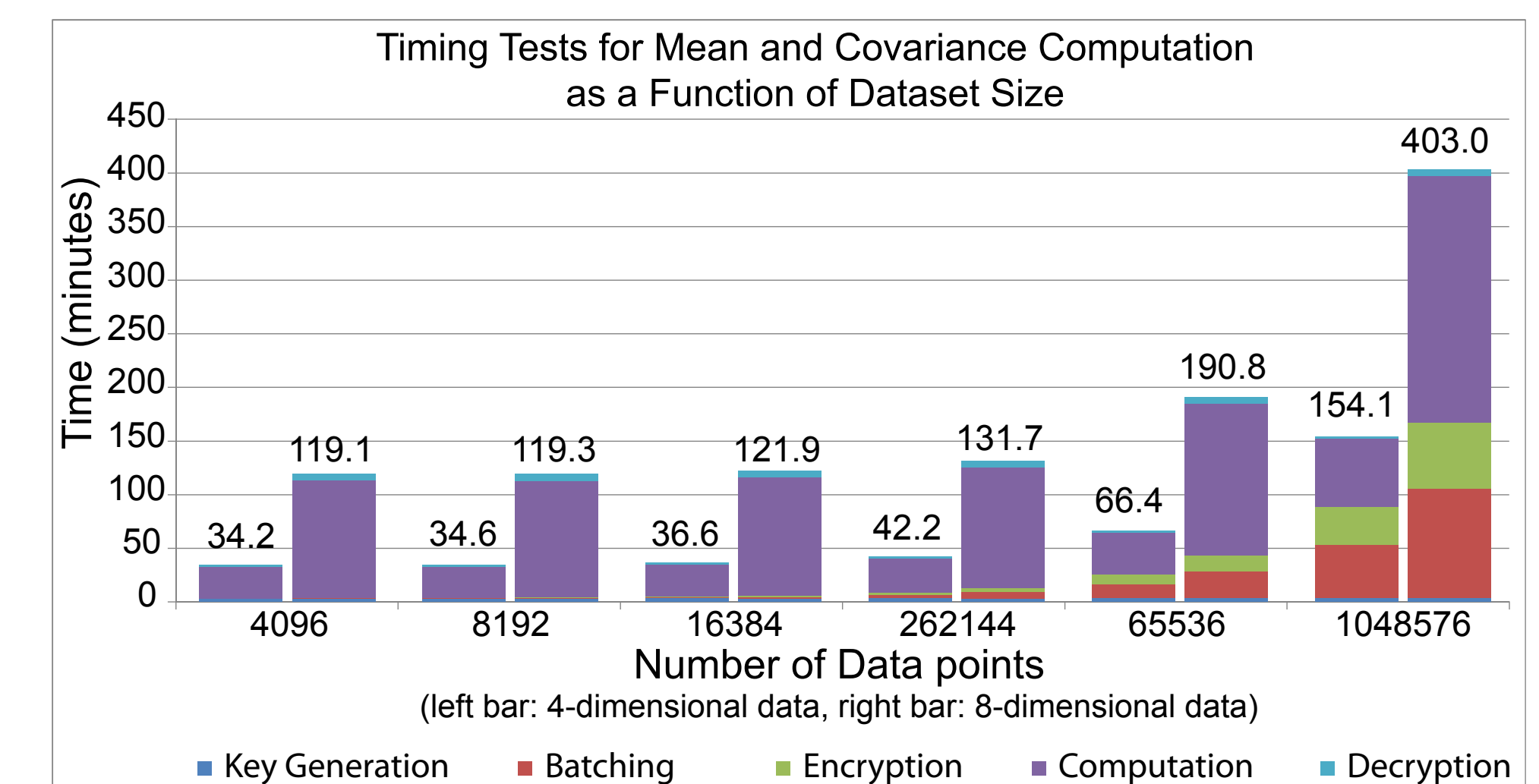
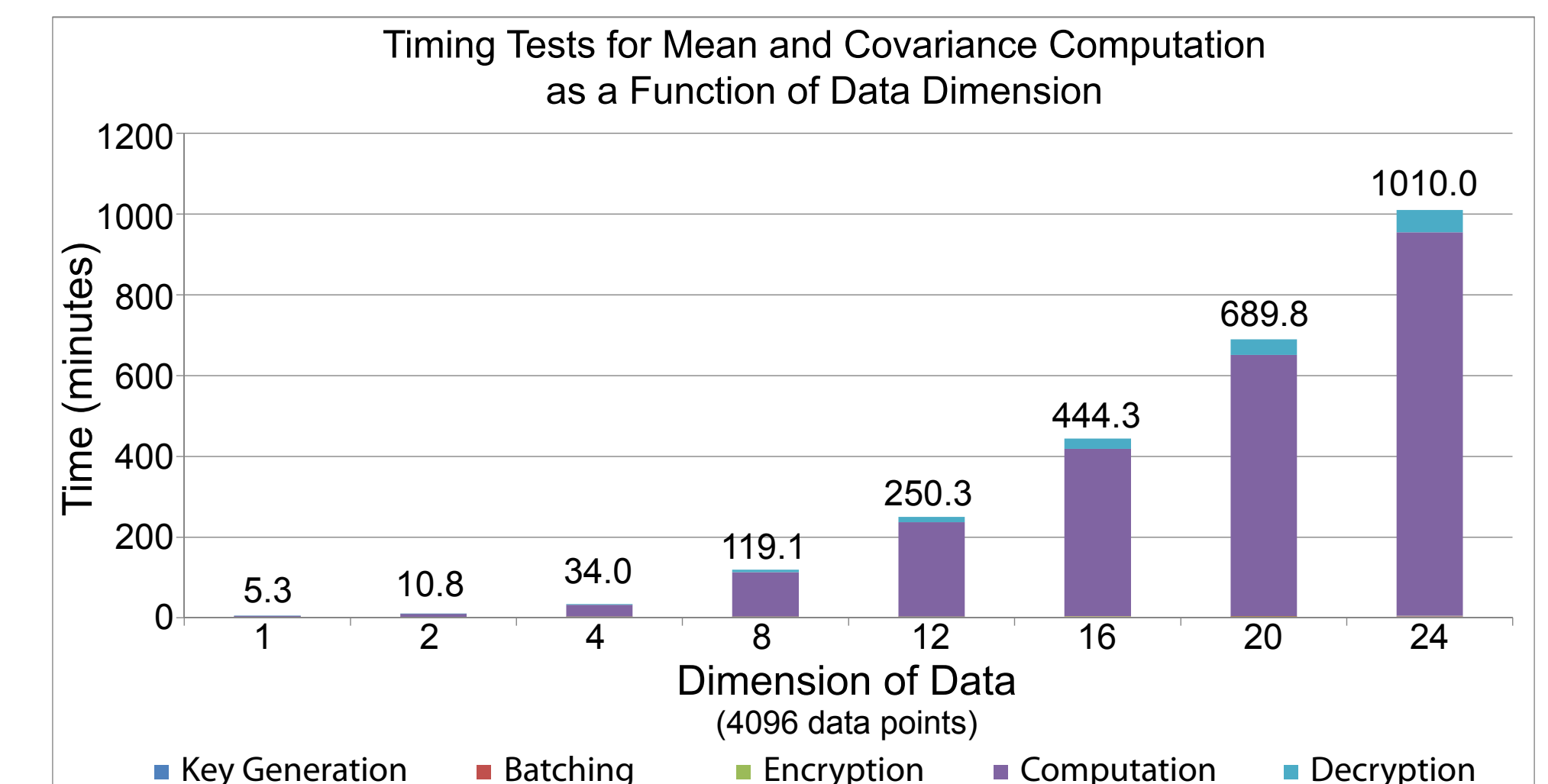
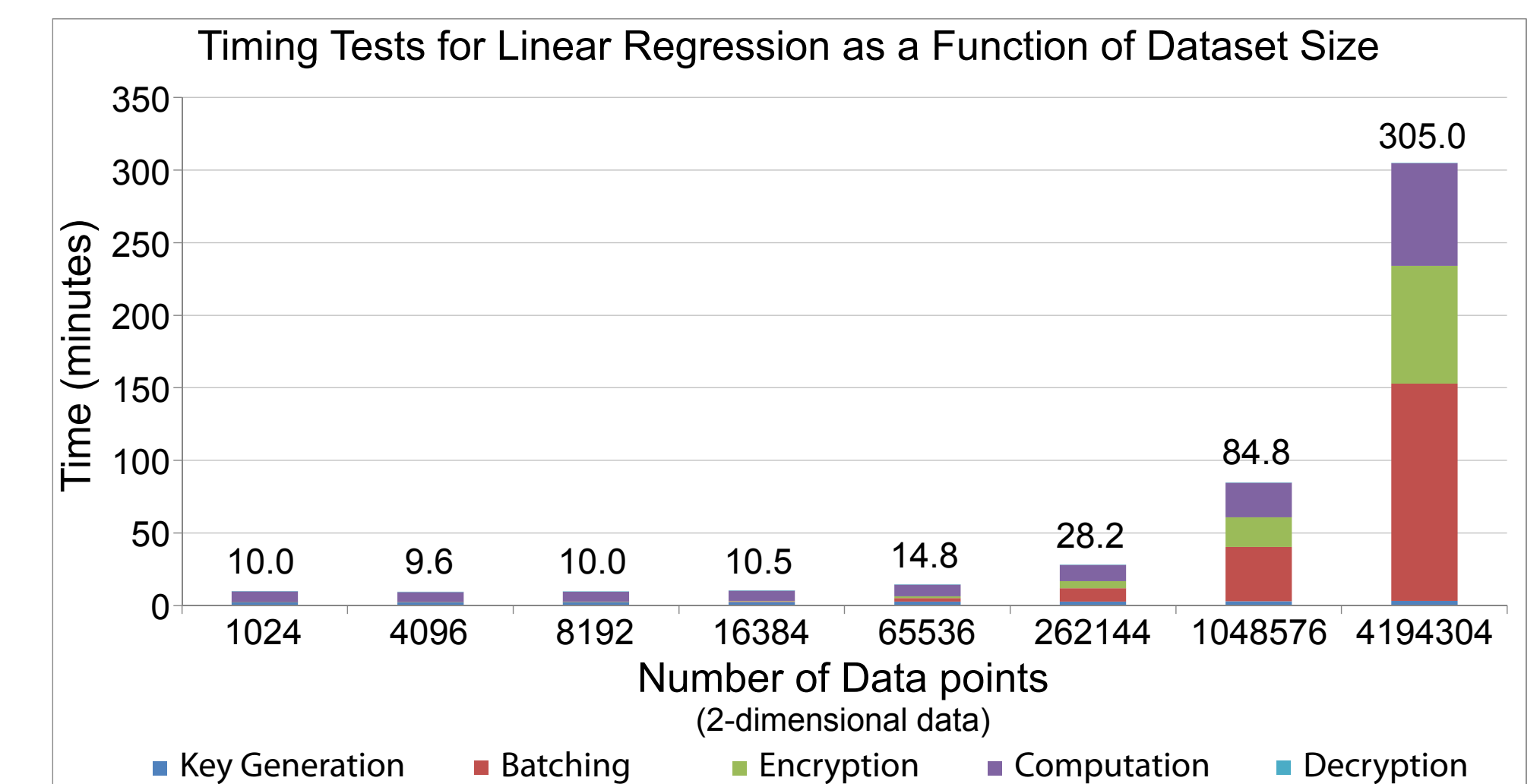
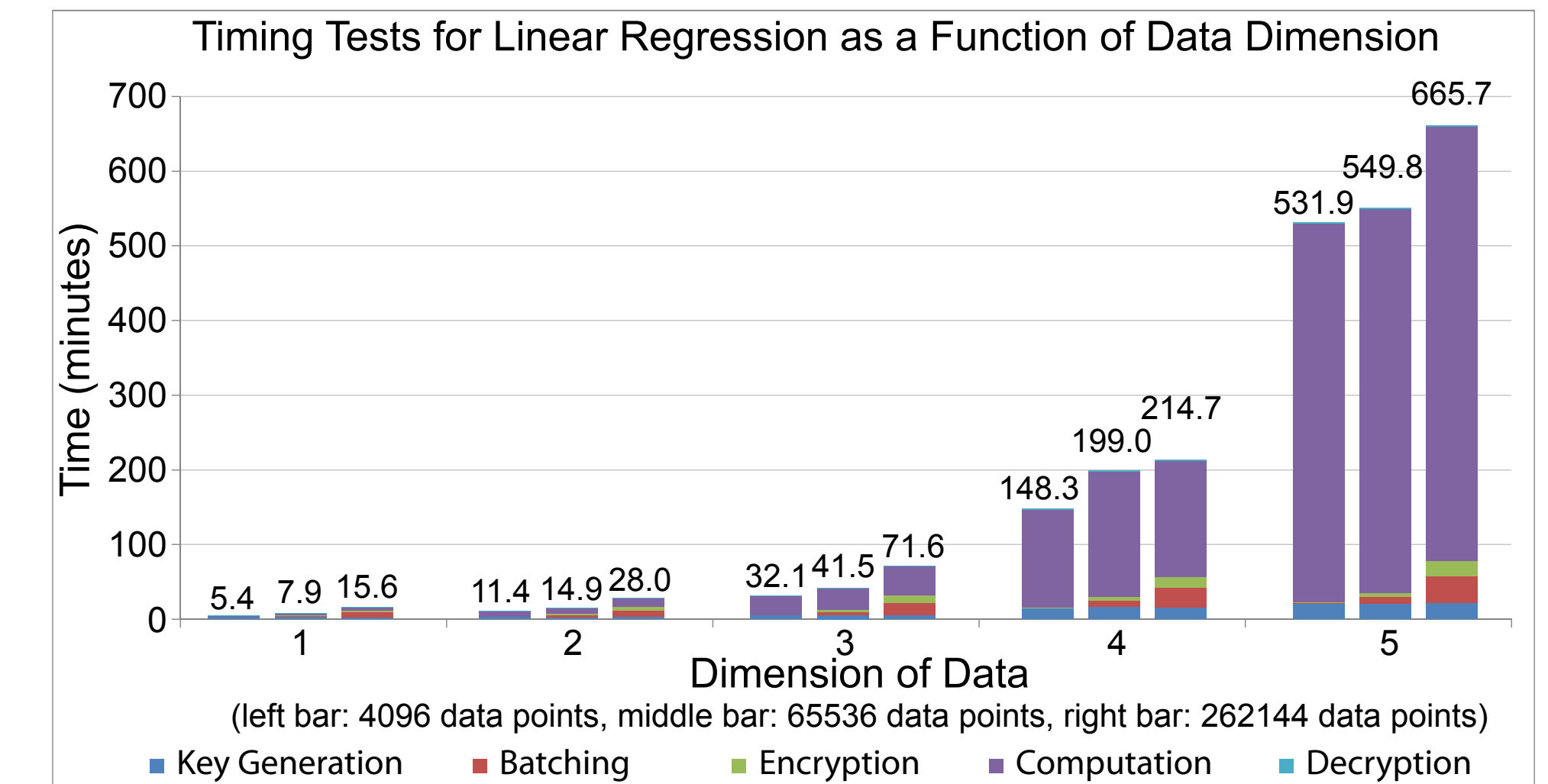
- \oplus Element-wise addition of batched ciphertexts.
- \otimes Element-wise multiplication of batched ciphertexts.
- κ_{g^i} Automorphism operation, which can apply arbitrary permutations to elements in the plaintext slots. Used here to rotate slots by i .

- Because the individual plaintext slots are non-interacting, we use a series of automorphisms to rotate the slots and additions to sum up the entries in a batch
- In the last step of the circuit, we zero out the values of the remaining slots. In the case where the number of slots is not a power of two, this ensures that no additional information about the data is leaked. The result is stored in the first slot of the final ciphertext.
- For security, we must add noise into ciphertexts during the encryption process. Homomorphic operations on ciphertexts increase this noise. In order to decrypt successfully, the noise must be below a chosen threshold.
- In fully homomorphic computation, multiplication is substantially more expensive (both in terms of runtime and amount of noise generated) than addition. We can quantify this by defining the *depth* of a circuit to be the number of multiplications in the circuit. Evaluating deeper circuits requires larger parameters, and correspondingly, longer runtimes. A comparison of addition and multiplication runtimes for different circuit depths is given below:

Depth	Time to Perform 1 Addition (ms)	Time to Perform 1 Multiplication (s)
1	1.94	0.62
2	3.23	2.14
5	10.81	14.11
10	25.44	102.20
20	141.93	771.55

Experiments

Below are results from timing tests illustrating performance of linear regression as well as mean and covariance computation on different datasets. Running times are relative to *one* prime in the CRT decomposition.



Conclusion

- We have constructed a scale-invariant leveled fully homomorphic encryption system.
- Using batching and CRT-based message encoding, we are able to perform *large scale* statistical analysis on millions of data points and data of moderate dimension.